34th International Symposium on Algorithms and Computation

ISAAC 2023, December 3-6, 2023, Kyoto, Japan

Edited by

Satoru Iwata

Naonori Kakimura

Editors

Satoru Iwata
University of Tokyo, Tokyo,
Hokkaido University, Sapporo, Japan
iwata@mist.i.u-tokyo.ac.jp

Naonori Kakimura
Keio University, Yokohama, Japan
kakimura@math.keio.ac.jp

ACM Classification 2012
Theory of computation; Mathematics of computing

ISBN 978-3-95977-289-1

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/978-3-95977-289-1.

Publication date
December, 2023

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.2023.0

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

34th International Symposium on Algorithms and Computation (ISAAC 2023).
Editors: Satoru Iwata and Naonori Kakimura
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the papers presented at the 34th International Symposium on Algorithms and Computation (ISAAC 2023), which was held in Kyoto, Japan on December 3–6, 2023, organized by Kyoto University, Japan. ISAAC 2023 provided a forum for researchers working in the areas of algorithms, theory of computation, and computational complexity. The technical program of the conference included 55 contributed papers. We received 184 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 55 of the submissions for presentation at the symposium.

The program committee selected the following papers as the recipients of the ISAAC 2023 Best Paper and Best Student Paper Awards.

- **Best Paper.** Carl Feghali, Felicia Lucke, Daniël Paulusma, and Bernard Ries: Matching Cuts in Graphs of High Girth and H-Free Graphs.
- **Best Paper.** Ragesh Jaiswal and Amit Kumar: Clustering What Matters in Constrained Settings: Improved Outlier to Outlier-Free Reductions.
- **Best Student Paper.** Nicola Cotumaccio: Prefix Sorting DFAs: a Recursive Algorithm.

The symposium included two invited presentations, delivered by Edith Elkind (University of Oxford, UK) and Seok-Hee Hong (University of Sydney, Australia). Abstracts of their talks are included in this volume. We are grateful to Inoue Foundation for Science, KDDI Foundation, Support Center for Advanced Telecommunications Technology Research (SCAT), The Telecommunications Advancement Foundation, and Algorithmic Foundations for Social Advancement (AFSA) by Grant-in-Aid for Transformative Research Areas, MEXT, Japan for financial support and to the local organizers of ISAAC 2023. Finally, we acknowledge the endorsement from Technical Committee on Theoretical Foundations of Computing (COMP) of IEICE and Special Interest Group on Algorithms (SIGAL) of IPSJ.

December 2023

Satoru Iwata and Naonori Kakimura

34th International Symposium on Algorithms and Computation (ISAAC 2023).
Editors: Satoru Iwata and Naonori Kakimura
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Hee-Kap Ahn (POSTECH, Korea)
Hyung-Chan An (Yonsei University, Korea)
Kevin Buchin (Technische Universität Dortmund, Germany)
Yixin Cao (Hong Kong Polytechnic University, China)
T-H. Hubert Chan (University of Hong Kong, China)
Karthekeyan Chandrasekaran (University of Illinois, Urbana-Champaign, USA)
Zhiyi Huang (University of Hong Kong, China)
Ayumi Igarashi (University of Tokyo, Japan)
Takehiro Ito (Tohoku University, Japan)
Satoru Iwata (Chair, University of Tokyo & Hokkaido University, Japan)
Taisuke Izumi (Osaka University, Japan)
Naonori Kakimura (Keio University, Japan)
Michael Lampis (University Paris Dauphine, France)
Euiwoong Lee (University of Michigan, USA)
Yi Li (Nanyang Technological University, Singapore)
Chung-Shou Liao (National Tsing Hua University, Taiwan)
Julian Mestre (University of Sydney, Australia)
Frédéric Meunier (École des Ponts, France)
Wolfgang Mulzer (Freie Universität Berlin, Germany)
Petra Mutzel (University of Bonn, Germany)
Alantha Newman (Université Grenoble Alpes, France)
Harumichi Nishimura (Nagoya University, Japan)
Eunjin Oh (POSTECH, Korea)
Laura Sanitá (Bocconi University, Italy)
Gregory Schwartzman (JAIST, Japan)
Kavitha Telikepalli (Tata Institute of Fundamental Research, India)
Seeun William Umboh (The University of Melbourne, Australia)
Chunhao Wang (Pennsylvania State University, USA)
Anthony Wirth (The University of Melbourne, Australia)
Wei Xu (Tsinghua University, China)
Yu Yokoi (Tokyo Institute of Technology, Japan)

34th International Symposium on Algorithms and Computation (ISAAC 2023).
Editors: Satoru Iwata and Naonori Kakimura

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

List of External Reviewers

Peyman Afshani
Taehoon Ahn
Marianne Akian
Hugo Akitaya
Tatsuya Akutsu
Xavier Allamigeon
Shinwoo An
Aditya Anand
Elena Arseneva
James Bailey
Niranka Banerjee
Paul Bell
Rémy Belmonte
Omri Ben-Eliezer
Ioana Bercea
Amey Bhangale
Umang Bhaskar
Sudatta Bhattacharya
Vijay Bhattiprolu
Davide Bilò
Jannis Blauth
Hans L. Bodlaender
Édouard Bonnet
Michaela Borzechowski
Valentin Bouquet
Cornelius Brand
Niv Buchbinder
Maike Buchin
Jin-Yi Cai
Clément Canonne
Matteo Ceccarello
Sankardeep Chakraborty
Yi-Jun Chang
Sudhanshu Chanpuriya
Abhranil Chatterjee
Ho-Lin Chen
Li-Hsuan Chen
Po-An Chen
Yurong Chen
Rajesh Chitnis
Kyungjin Cho
Keerti Choudhary
Chaeyoon Chung
Jaehoon Chung
Jonas Cleve
Nicola Cotumaccio
Radu Curticapean
Nathael Da Costa
Syamantak Das
Joshua Daymude
Jean-Lou De Carufel
Ronald de Haan
Paloma de Lima
Arnaud De Mesmay
Fabien De Montgolfier
Claire Delaplace
Argyrios Deligkas
Dong Deng
Patrick Eades
Martijn van Ee
Ryota Eguchi
Friedrich Eisenbrand
Jonas Ellert
Matthias Englert
Taekang Eom
Joshua Erde
Andreas Emil Feldmann
Weiming Feng
Yiding Feng
Pamela Fleischmann
Till Fluschnik
Klaus-Tycho Foerster
Mathew Francis
Takuro Fukunaga
Junhao Gan
Serge Gaspers
Colin Geniet
Daniel Gibney
Tatsuya Gima
Andreas Göbel
Adrián Goga
Petr Golovach
Alexander Golovnev
Fabrizio Grandoni
Yan Gu
Tesshu Hanaka
Ararat Harutyunyan
Koyo Hayashi
Kun He
Meng He

34th International Symposium on Algorithms and Computation (ISAAC 2023).
Editors: Satoru Iwata and Naonori Kakimura
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Klaus Heeger
Kevin Hendrey
Duc A. Hoang
Alexandros Hollender
Wing-Kai Hon
Lingxiao Huang
Shenwei Huang
Zijin Huang
Ling-Ju Hung
Yuni Iwamasa
Afrouz Jabal Ameli
Lars Jaffke
Stephen Jaud
Shaofeng Jiang
Yonggang Jiang
Seungbum Jo
Mook Kwon Jung
Dominik Kaaser
Byeonguk Kang
Alexander Kauer
Jun Kawahara
Yasushi Kawase
Safia Kedad-Sidhoum
Arindam Khan
Eun Jung Kim
Hwi Kim
Yonghwan Kim
Sándor Kisfaludi-Bak
Boris Klemz
Katharina Klost
Fabian Klute
Kristin Knorr
Sang-Ki Ko
Yasuaki Kobayashi
Yusuke Kobayashi
Noleen Köhler
Tuukka Korhonen
Laszlo Kozma
Alexander Kulikov
Shubhang Kulkarni
Pankaj Kumar
O-Joung Kwon
Elmar Langetepe
Hoang-Oanh Le
Hung Le
Francois Le Gall
Chui Shan Lee
Jaegun Lee
Seungjun Lee
Jianqiang Li
Meng-Hsi Li
Shi Li
Young-San Lin
Andrzej Lingas
Kelin Luo
Jayson Lynch
Shunichi Maezawa
Ryoga Mahara
Florin Manea
Pasin Manurangsi
Giovanni Manzini
Nikolaos Melissinos
Yang Meng
Arturo Merino
Takuya Mieno
Pranabendu Misra
Shuichi Miyazaki
Benjamin Moseley
Guiqiang Mou
Moritz Muehlenthaler
Yonatan Nakar
Yuto Nakashima
Jesper Nederlof
Gabriel Nivasch
André Nusser
Naoto Ohsaka
Yoshio Okamoto
Taihei Oki
Lukáš Ondráček
Fukuhito Ooshita
Michal Opler
Joachim Orthaber
Yota Otachi
Katarzyna Paluch
Ami Paz
Binghui Peng
Asaf Petruschka
Marc Pfetsch
Matthias Pfretzschner
Christophe Picouleau
Vladimir Podolskii
Germain Poullot
Amaury Pouly
Pegah Pournajafi
Gautam Prakriya
Nicola Prezza

Saladi Rahul
Ashutosh Rai
Malin Rau
Carolin Rehs
André van Renssen
Adele Rescigno
Bernard Ries
Heiko Röglin
Thomas Rothvoss
Eric Rowland
Bodhayan Roy
Kunihiko Sadakane
Toshiki Saitoh
Yoshifumi Sakai
Sherry Sarkar
Maria Saumell
Ildikó Schlotter
Lia Schütze
Chris Schwiegelshohn
Andras Sebo
Martin P. Seybold
Yuan Sha
Amatya Sharma
Gokarna Sharma
Takeharu Shiraga
Igor Shparlinski
Xinkai Shu
Sebastian Siebertz
Bertrand Simon
Sunil Simon
Kirill Simonov
Mohit Singh
Shikha Singh
Nodari Sitchinava
Minju Song
Francisco Soulignac
K. Subramani
Yuichi Sudo
Noriyoshi Sukegawa
Hanna Sumita
Enze Sun
Yiming Sun
Akira Suzuki
Tsuyoshi Takagi
Mizuyo Takamatsu
Kenjiro Takazawa
Toru Takisaka
Suguru Tamaki
Yuma Tamura
Jakub Tarnawski
Pablo Torres
Meng-Tsung Tsai
Garnero Valentin
Manolis Vasilakis
Daniel Vaz
Giovanni Viglietta
Magnus Wahlström
Haitao Wang
Hung-Lung Wang
Shenghua Wang
Kunihiro Wasa
Sebastian Wiederrecht
Max Willert
Michal Wlodarczyk
Petra Wolf
Sampson Wong
Pei Wu
Mingji Xia
Chao Xu
Luze Xu
Yinzhan Xu
Jie Xue
Yutaro Yamaguchi
Yukiko Yamauchi
Wei Yu
Yang Yuan
Ahad N. Zehmakan
Chihao Zhang
Guochuan Zhang
Peng Zhang
Xinyuan Zhang
Xinzhi Zhang
Yuhao Zhang
Yuxuan Zhang
Rudy Zhou
Isabella Ziccardi

Group Fairness: From Multiwinner Voting to Participatory Budgeting

Edith Elkind ✉
University of Oxford, UK
Alan Turing Institute, London, UK

Abstract

Many cities around the world allocate a part of their budget based on residents' votes, following a process known as participatory budgeting. It is important to understand which outcomes of this process should be viewed as fair, and whether fair outcomes could be computed efficiently. We summarise recent progress on this topic. We first focus on a special case of participatory budgeting where all candidate projects have the same cost (known as multiwinner voting), formulate progressively more demanding notions of fairness for this setting, and identify efficiently computable voting rules that satisfy them. We then discuss the challenges of extending these ideas to the general model.

2012 ACM Subject Classification Applied computing

Keywords and phrases multiwinner voting, participatory budgeting, justified representation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.1

Category Invited Talk

1 Talk Summary

Many cities around the world allocate a fraction of their budget using a process known as *participatory budgeting*: the local authority decides on a budget, collects proposals for projects to be implemented (such as, e.g., renovating a playground, building an outdoor gym, or planting trees), vets them for suitability, comes up with a cost estimate for each candidate project, and then elicits the residents' preferences over these projects [5, 11, 3]. The ballots are then aggregated to decide which subset of projects should be implemented. The goal of the aggregation process is to select a set of projects that fits within the given budget, yet reflects the residents' desires.

The implementation details of preference elicitation and aggregation processes vary from one municipality to another. Some localities ask the voters to rank the projects or to approve the projects they like (sometimes there is a bound on the number of projects that one is allowed to approve), while others ask them to specify their "ideal" budget allocation (see, e.g., the discussion in the work of Benade et al. [4]). Then, once the preferences are submitted, there is a multitude of procedures that can be used to aggregate the ballots into a collective decision. For instance, if voters are required to submit approval ballots, i.e., list all projects that they approve, the simplest (and widely used) method is to use the greedy strategy: one can order all projects either by the number of approvals they receive or by the number of approvals divided by the cost of the projects ('bang for the buck'), and add the projects to the selection one by one, skipping over the projects if adding them would violate the budget constraint.

However, the greedy approach may result in outcomes that are clearly undesirable. Imagine, for instance, that a city is partitioned into two districts by the railroad tracks, with 300,000 people living south of the tracks and 280,000 people living north of the tracks. As there are few track crossings, the residents of each district only approve projects in their own district. Now, suppose the overall budget is \$1,000,000, and in each district there

are five popular proposals that cost \$200,000 each to implement (as well as various other less popular proposals). Then, if 60% of the residents of each district participate in the selection process, all five south-of-the-tracks popular projects receive more votes than all five north-of-the-tracks projects, so the greedy approach would result in all money being spent according to the wishes of the south-of-the-tracks district residents. Obviously, a much fairer solution would be to implement three projects in the south and two projects in the north.

One can ensure fair distribution of funding across geographic areas by explicitly dividing the budget among the districts in proportion to their population, but geographic diversity is not the only concern that we need to worry about. For instance, we do not want the entire budget to be spent on cycling infrastructure, or on children's facilities; indeed, we want our decisions to be fair to all groups of residents, including those that the city council is not aware of. Conceptually, one can argue that if the total budget is B and the population size is n, each resident should be entitled to allocate B/n dollars in any way they wish. This idea, which is closely related to the concept of the core on cooperative game theory, suggests a decentralised procedure, where residents can form groups to jointly support various projects. However, the majority of residents would probably find it too burdensome to engage in negotiations. Is there a simple voting rule that provides the same guarantees, but can be implemented by the city council based on the residents' ballots?

In this survey, we summarise the state of the art regarding participatory budgeting with group fairness guarantees, under the assumption that voters' preferences are captured by approval ballots (i.e., each voter approves a subset of projects, and is indifferent among all projects they approve). We formulate several axioms that aim to capture what it means for a budget allocation process to be fair, describe voting rules that satisfy (some of) these axioms, and discuss their algorithmic complexity.

In more detail, we start by considering the setting of multiwinner voting, i.e., a special case of participatory budgeting where all projects have the same cost. We formulate several fairness axioms for this model, such as justified representation [1], proportional justified representation [10], extended justified representation [1], full justified representation [8], and the core [1]. We present the definitions of popular multiwinner voting rules, focusing, in particular, on Proportional Approval Voting (PAV) [6], the recently introduced Method of Equal Shares [9], and their variants [2], and discuss their computational complexity. We then discuss generalisations of these axioms and voting rules to the setting of participatory budgeting, and outline their strengths and limitations [9, 7].

References

1 Haris Aziz, Markus Brill, Vincent Conitzer, Edith Elkind, Rupert Freeman, and Toby Walsh. Justified representation in approval-based committee voting. *Social Choice and Welfare*, 48(2):461–485, 2017.

2 Haris Aziz, Edith Elkind, Shenwei Huang, Martin Lackner, Luis Sánchez-Fernández, and Piotr Skowron. On the complexity of extended and proportional justified representation. In *AAAI'18*, 2018.

3 Haris Aziz and Nisarg Shah. Participatory budgeting: Models and approaches. *Pathways Between Social Science and Computational Social Science: Theories, Methods, and Interpretations*, pages 215–236, 2021.

4 Gerdus Benade, Swaprava Nath, Ariel D Procaccia, and Nisarg Shah. Preference elicitation for participatory budgeting. *Management Science*, 67(5):2813–2827, 2021.

5 Yves Cabannes. Participatory budgeting: a significant contribution to participatory democracy. *Environment and urbanization*, 16(1):27–46, 2004.

6 D. Marc Kilgour. Approval balloting for multi-winner elections. In *Handbook on Approval Voting*, pages 105–124. Springer, 2010.

7 Sonja Kraiczy and Edith Elkind. An adaptive and verifiably proportional method for participatory budgeting. In *WINE'23*, 2023.

8 Dominik Peters, Grzegorz Pierczyński, and Piotr Skowron. Proportional participatory budgeting with additive utilities. In *NeurIPS'21*, pages 12726–12737, 2021.

9 Dominik Peters and Piotr Skowron. Proportionality and the limits of welfarism. In *ACM EC'20*, pages 793–794, 2020.

10 Luis Sánchez-Fernández, Edith Elkind, Martin Lackner, Norberto Fernández, Jesús Fisteus, Pablo Basanta Val, and Piotr Skowron. Proportional justified representation. In *AAAI'17*, pages 670–676, 2017.

11 Brian Wampler, Stephanie McNulty, and Michael Touchton. *Participatory budgeting in global perspective*. Oxford University Press, 2021.

Faithful Graph Drawing

Seok-Hee Hong
School of Computer Science, The University of Sydney, Australia

Abstract

Graph drawing aims to compute good geometric representations of graphs in two or three dimensions. It has wide applications in network visualisation, such as social networks and biological networks, arising from many other disciplines.

This talk will review fundamental theoretical results as well as recent advances in graph drawing, including symmetric graph drawing, generalisation of the Tutte's barycenter theorem, Steinitz's theorem, and Fáry's theorem, and the so-called *beyond planar* graphs such as k-planar graphs.

I will conclude my talk with recent progress in visualization of big complex graphs, including sublinear-time graph drawing algorithms and *faithful* graph drawing.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graph drawing, Planar graphs, Beyond planar graphs, Tutte's barycenter theorem, Steinitz's theorem, Fáry's theorem, Sublinear-time graph drawing algorithm, Faithful graph drawing, Symmetric graph drawing

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.2

Category Invited Talk

Funding Research supported by ARC (Australian Research Council) Future Fellowship and ARC Discovery Projects.

Acknowledgements I would like to thank all my research collaborators, in particular, Peter Eades, Hiroshi Nagamochi, Giuseppe Liotta, Amyra Meidiana, and Weidong Huang.

Realizability of Free Spaces of Curves

Hugo A. Akitaya
Department of Computer Science, University of Massachusetts Lowell, MA, USA

Maike Buchin
Department of Computer Science, Ruhr University Bochum, Germany

Majid Mirzanezhad
Transportation Research Institute, University of Michigan, Ann Arbor, MI, USA

Leonie Ryvkin
Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands

Carola Wenk
Department of Computer Science, Tulane University, New Orleans, LA, USA

Abstract

The free space diagram is a popular tool to compute the well-known Fréchet distance. As the Fréchet distance is used in many different fields, many variants have been established to cover the specific needs of these applications. Often the question arises whether a certain pattern in the free space diagram is *realizable*, i.e., whether there exists a pair of polygonal chains whose free space diagram corresponds to it. The answer to this question may help in deciding the computational complexity of these distance measures, as well as allowing to design more efficient algorithms for restricted input classes that avoid certain free space patterns. Therefore we study the inverse problem: Given a potential free space diagram, do there exist curves that generate this diagram?

Our problem of interest is closely tied to the classic Distance Geometry problem. We settle the complexity of Distance Geometry in $\mathbb{R}^{>2}$, showing $\exists\mathbb{R}$-hardness. We use this to show that for curves in $\mathbb{R}^{\geq 2}$ the realizability problem is $\exists\mathbb{R}$-complete, both for continuous and for discrete Fréchet distance. We prove that the continuous case in $\mathbb{R}^1$ is only weakly NP-hard, and we provide a pseudo-polynomial time algorithm and show that it is fixed-parameter tractable. Interestingly, for the discrete case in $\mathbb{R}^1$ we show that the problem becomes solvable in polynomial time.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Fréchet distance, Distance Geometry, free space diagram, inverse problem

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.3

Related Version *Full Version*: https://arxiv.org/abs/2311.07573

Funding *Carola Wenk*: partially supported by NSF grant CCF 2107434.

1 Introduction

The Fréchet distance is arguably the most popular distance measure for curves in computational geometry and has been studied extensively in the last years. It has application in various fields, including geographic data analysis and the comparison of protein chains [25, 26, 35, 28]. For the latter application, typically the well-established variant, the discrete Fréchet distance, is used. The standard tool for computing the Fréchet distance of two curves is the *free space diagram*, which is the cross-product of the parameter spaces of the curves partitioned into *free space* and its complement, where free space is the sublevel set of the distance function for a given $\varepsilon > 0$. For two piecewise linear curves of m and n line segments parameterized by their natural arc-length parametrization, it is well-known that the free space diagram consists of mn cells, and the free space in each cell has the shape of a cropped ellipse [5]. The

Fréchet distance is at most ε if and only if there exists a monotone path in the free space diagram that covers the parameter spaces of both curves. Hence, to compute the Fréchet distance one searches for such a path in the free space diagram.

For different applications, many variants of the Fréchet distance have been developed, which are typically also computed using the free space diagram. The discrete Fréchet distance relies on (a discretization of the free space diagram) the *free space matrix*: For two discretized curves given by n and m points, respectively, the $n \times m$ free space matrix with entries $a_{i,j} \in \{0, 1\}$ captures whether two points i, j of different curves lie within $\varepsilon > 0$ distance of one another, or not. The discrete Fréchet distance of two curves is at most ε iff there exists a monotone "path" of 1 entries connecting opposing corners in the free space matrix.

Runtimes of the resulting algorithms usually depend on the complexity of the free space diagram or free space matrix [23, 8, 10]. It is known that neither Fréchet distance nor discrete Fréchet distance can be computed in subquadratic time unless SETH fails [12, 16, 13]. However, there are several faster algorithms for special curve classes such as [6, 24], which exploit a special structure of the free space diagram. These complexity bounds always consider the worst-case complexity of the free space diagram, but not every free space diagram can be realized by a pair of curves. Also, some variants of the Fréchet distance have proven to be NP-hard to decide [4, 17], some of which build certain free space diagrams for the reduction.

Here we therefore study the inverse problem: Given a (potential) free space diagram (free space matrix), do there exist curves (ordered point sets) that generate this free space diagram or matrix? To our knowledge this *free space realizability problem* has so far only been studied for special cases [32, 18, 3]. Understanding it will give structural insights into free spaces and the computation of the Fréchet distance, in particular for special curve classes. Note that although we gained a good understanding of the realizability problem in the settings described below, other settings remain to be investigated further. We are particularly interested in studying settings where less information is given in the free space diagram or matrix, e.g. only some cells or only cell boundaries are provided with the input.

Overview. We give results (see Table 1) for both the continuous and the discrete variant of the problem, with free space diagram and free space matrix inputs, and curves may be realized in $\mathbb{R}^d$, with $d = 1$ or $d \geq 2$. We show that for curves in $\mathbb{R}^{\geq 2}$ the realizability problem is $\exists\mathbb{R}$-complete both for the continuous case (Section 3) and for the discrete case (Section 5). For the continuous case in $\mathbb{R}^2$, algorithms are known only for special cases [32, 18]. For curves in $\mathbb{R}^1$, the problem in the discrete case interestingly becomes solvable in polynomial time (Section 6), while in the continuous case, it is weakly NP-hard (Section 4) and fixed-parameter tractable, and we also provide a pseudo-polynomial time algorithm (Section 4).

Table 1 Overview of our and known results.

Input	$\mathbb{R}^d$	Results
Free Space Diagram	$d \geq 2$	$\exists\mathbb{R}$-complete (Theorem 4) **Section 3** Algorithms for special cases ([32, 18])
	$d = 1$	weakly NP-hard (Theorem 7) **Section 4** FPT $O(mn2^k)$ (Theorem 8) **Section 4** Pseudo-poly time (Theorem 13) **Section 4**
Free Space Matrix	$d \geq 2$	$\exists\mathbb{R}$-complete (Theorem 14) **Section 5**
	$d = 1$	$O(nm^2)$ (Theorem 16) **Section 6**

Related problems. The exploration of inverse problems is a recurrent subject in computational geometry, often applied to recognition and reconstruction problems. Notable examples are the *inverse Voronoi Diagram* problem [9] and the *visibility graph recognition* problem [11]. Our problem of interest can be viewed as a curve embeddability problem given certain proximity criteria on edge lengths and point-to-point distances between two curves in their free space diagram. Our results have a significant impact on problems involving distance constraints on geometric graphs, such as *Distance Geometry* and *Disk Intersection Graphs*.

Distance Geometry. Our results closely relate to problems involving distance constraints on geometric graphs, such as the *Distance Geometry* problem, which is a classic inverse problem in computational geometry. It involves embedding an abstract weighted graph in $\mathbb{R}^d$ Euclidean space such that the Euclidean length of each edge corresponds to its weight [33]. This problem is equivalent to *Linkage Realizability*, where the edges correspond to rigid bars in a mechanical linkage [33]. While distance geometry was shown to be NP-hard in the 70s, its membership in NP remained open until Schaefer showed $\exists\mathbb{R}$-completeness in 2012 [34] for $\mathbb{R}^2$. Whether the problem remains $\exists\mathbb{R}$-hard in higher dimensions was posed as an open problem by Schaefer. To show that the continuous version of the free space realizability problem in $\mathbb{R}^{\geq 2}$ is $\exists\mathbb{R}$-hard we reduce from distance geometry, using a gadget from [33].

Unit Sphere Graphs. The sphericity of a graph is the minimum dimension for which the graph has a unit sphere representation. Unit sphere graph realizability is known to be $\exists\mathbb{R}$-hard. Havel first introduced the study of sphericity in the context of molecular conformation [27]. A unit sphere graph is an intersection graph of unit spheres and can be seen as a complete graph where each edge is marked with a distance constraint ≤ 1 or > 1. The problem of realizing a free space matrix corresponds to realizing a complete bipartite graph where each edge is marked with a distance constraint ≤ 1 or > 1 (Section 5). This defines a class of graphs, as do unit disk graphs. In contrast to a unit disk graph, there are pairs of vertices (the ones in the same partite set) whose distances are dispensable. A similar class are visibility graphs, the recognition of which is also known to be $\exists\mathbb{R}$-complete [19].

Bipartite Distance-Constrained Graphs. Modeling distance constraints in general (non-complete) graphs is useful when data is unavailable between every pair of nodes or due to the topology of the underlying network. E.g., heteronuclear NMR is used to obtain less cluttered and less noisy data [30]. This allows the inference of distances between two different types of atoms, and thus, the distances constraints form a bipartite graph.

2 Preliminaries

Continuous case. Let $P = (p_0, \ldots, p_n)$ and $Q = (q_0, \ldots, q_m)$ be polygonal curves in $\mathbb{R}^d$ of lengths ℓ_P and ℓ_Q, continuously parameterized by arc-length, i.e. $p_i = P(i)$ and $q_j = Q(j)$ for $i \in [n]$, $j \in [m]$, where $[n] = \{1, \cdots, n\}$. Given $\varepsilon > 0$, their *free space* is defined as $F_\varepsilon(P, Q) = \{(r, t) \mid \|P(r) - Q(t)\| \leq \varepsilon\}$. The *free space diagram* puts this information in an $m \times n$ grid: We define $D_\varepsilon(P, Q)$ as the colored rectangle $R = [0, \ell_P] \times [0, \ell_Q] \subseteq \mathbb{R}^2$, where a point $(p, q) \in R$ is colored white iff $(p, q) \in F_\varepsilon(P, Q)$. The grid X, which is the set of segments $\{p_i \times [0, \ell_Q] \mid i \in \{0, \ldots, n\}\} \cup \{[0, \ell_P] \times q_j \mid j \in \{0, \ldots, m\}\}$, subdivides R into $n \times m$ cells $C_{i,j}$. We call a single cell $C_{i,j}$ *empty* if $C_{i,j} \cap F_\varepsilon = \emptyset$ and *full* if $C_{i,j} \cap F_\varepsilon = C_{i,j}$. If $\emptyset \neq C_{i,j} \cap F_\varepsilon \neq C_{i,j}$, the cell $C_{i,j}$ is called *partially full*. A diagram D_ε is called *realizable* if there exist curves P, Q such that $D_\varepsilon(P, Q) = D_\varepsilon$. We assume that the exact lengths of

all cell boundaries and the equation of each component's boundary curve are part of the input. We consider the real RAM computation model throughout the paper. In Figure 1, the leftmost diagram is not realizable: Upon fixing the placement of segments corresponding to cell $C_{1,1}$, we have two options to place the remaining segments such that cell $C_{2,2}$ is realized. This either induces components in none of the remaining cells, or in both.

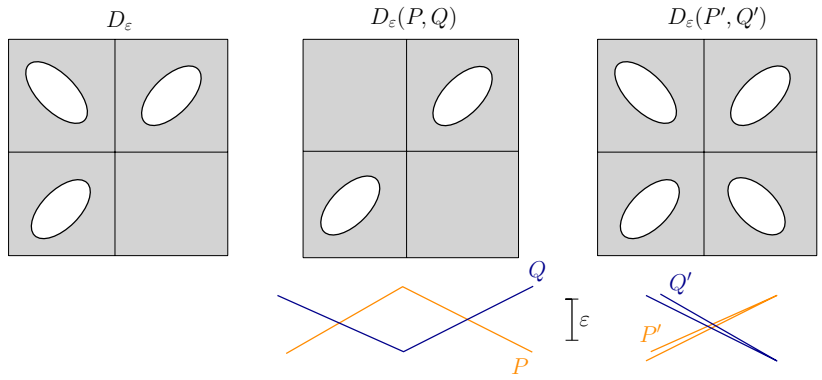


Figure 1 Given diagram D_ε, there are two ways to place curves P, Q in $\mathbb{R}^2$. Neither realizes D_ε.

In the following, we denote the ball of radius r centered at a point $x \in \mathbb{R}^d$ as $\mathcal{B}_r(x) := \{p \in \mathbb{R}^d \mid \|x - p\| \leq r\}$. The *$\varepsilon$-neighborhood* of an object X is given by $\bigcup_{x \in X} \mathcal{B}_\varepsilon(x)$. We denote by $s_i^P = \overline{p_{i-1}p_i}$, $i \in [n]$, the line segment connecting consecutive vertices of P, and define s_i^Q analogously. In $\mathbb{R}^1$, if two consecutive segments s_i^P, s_{i+1}^P have different orientations (segments are placed on top of each other), we say the curve *folds* at the common *folding vertex* p_i. Else, we say that the curve is *straight* at p_i. It is known that the free space of two lines has the shape of a cropped ellipse with axis at $\pm 45°$ [5, 31]. For curves in $\mathbb{R}^1$, the lines are necessarily parallel, hence the ellipse degenerates to a slab bounded by lines at $\pm 45°$ [14].

Partially full cells. We now establish a necessary condition for curves P, Q to realize D_ε that is used in Sections 3–4. Partially full cells give us information about the relative placement of the segments. We use the term *relative placement* of two segments to mean that we know the distances between the intersection point of the lines containing the segments and the segment endpoints. Note that after fixing the position of one segment, this still allows for two symmetric placements of the second segment.

▶ **Lemma 1.** *Given a partially full free space cell, $\varepsilon > 0$, and four points on the boundary of the ellipse in the cell, none of which are mirror images of another with respect to the ellipse's major and minor axes, we can compute the corresponding segments' relative placement.*

The proof (see Lemma 5.5 in [32]) relies on knowing that a cell is an ellipse at 45°. Note that we obtain much less information from full or empty cells, namely only that the segments do or do not lie within or not within distance ε from each other.

Definitions: Discrete case. For discrete polygonal curves (i.e., point sequences) P, Q with n and m points, resp., the free space is defined as $F_\varepsilon(P,Q) = \{(i,j) \in [n] \times [m] \mid \|p_i - q_j\| \leq \varepsilon\}$. We define the free space matrix $M_\varepsilon(P,Q)$ as the $n \times m$ matrix featuring entries $a_{i,j} \in \{0,1\}$, $i \in [n]$, $j \in [m]$ where $a_{i,j} = 1$ if and only if $(i,j) \in F_\varepsilon(P,Q)$. For a given matrix M_ε we ask whether there exist curves P, Q such that $M_\varepsilon = M_\varepsilon(P,Q)$.

3 $\exists\mathbb{R}$-Completeness for Continuous Curves

We first show that given a diagram D_ε, the problem of finding two curves in $\mathbb{R}^2$ that realize D_ε is $\exists\mathbb{R}$-complete. We then generalize to higher dimensions in Section 3.1. Containment in $\exists\mathbb{R}$ is shown by expressing the problem using real inequalities, see Lemma 17, Appendix A.

We reduce from the problem of deciding whether a linkage has a planar realization which was shown $\exists\mathbb{R}$-hard by Abel et al. [1, 2]. A *mechanical linkage* is a mechanism made of rigid bars connected at hinges. The input is a weighted graph $G = (V(G), E(G), \ell_G)$, where ℓ_G is the weight function, and a function $\Pi\colon W \to \mathbb{R}^2$, where $W \subseteq V(G)$, that represents vertices whose positions are *pinned*. A *configuration* C of a linkage $\mathcal{L} = (G, \Pi)$ is a straight-line drawing of G where the length of each edge $e \in E(G)$ is $\ell_G(e)$ and the position of each vertex $w \in W$ is $\Pi(w)$. The linkage realization problem asks whether a given linkage admits a configuration. A configuration C is *noncrossing* if C is a plane drawing. Abel et al. [1, 2] showed that the linkage realization problem remains hard for a series of restrictions on the input linkage $\mathcal{L}$. We restate a direct consequence of Theorems 2.2.13 and 2.4.6 in [2]. Although not all conditions in the theorem below are explicitly stated in [2, Theorem 2.2.13], they can be directly inferred by their construction in [2, Section 2.7].

▶ **Theorem 2** (Simplified from [2], Theorems 2.2.13 and 2.4.6)**.** *Given a linkage $\mathcal{L} = (G, \Pi)$ and a combinatorial embedding (clockwise circular order of edges around each vertex) σ of G, deciding whether there exists a planar realization of $\mathcal{L}$ is $\exists\mathbb{R}$-hard even if the following constraints are enforced:*

1. *G is connected, and the length of every edge is an integer.*
2. *A set of edge disjoint subgraphs H of G can be assigned* rigid*, i.e., each angle between consecutive incident edges in H is prescribed from $\{90°, 180°, 270°, 360°\}$. Each subgraph H is a tree, and an edge in $E(G) \setminus E(H)$ incident to H must be incident to a leaf of H.*
3. *Only three vertices are pinned ($|\Pi| = 3$), all three belong to the same rigid subgraph H (described in constraint (2)), and they are not collinear.*
4. *For every noncrossing configuration C of $\mathcal{L}$ that satisfies constraints (1–3), holds:*
   a. *C agrees with σ.*
   b. *Angles that are not prescribed by constraint (2) lie strictly between $60°$ and $240°$.*
   c. *The minimum distance of a vertex and a nonincident edge is at least a constant ϕ.*

We call a vertex *rigid* if it is incident to at least two edges of the same rigid subgraph H, and *nonrigid* otherwise. By (2), every angle incident to a rigid vertex is prescribed while no angle in a nonrigid vertex is prescribed (which by (4b) can only vary in the interval $(60°, 240°)$). Note that distance geometry is equivalent to linkage realization with $\Pi = \emptyset$. Since (3) makes Π irrelevant, Theorem 2 also implies hardness for distance geometry.

Reduction description. Given $\mathcal{L}$ and σ satisfying the constraints in Theorem 2, we construct an instance D_ε as follows. A full example can be seen in Figure 2. The idea is to build a free space such that realizing curves trace out the linkage, following the given combinatorial embedding. For this, we first transform G into a tree. The angle constraints in the linkage can also be enforced in the free space using a specific gadget.

While there is a cycle in G, split one edge in a cycle by placing a new vertex in its midpoint and performing a vertex split, creating two copies of the new vertex, each attached to half of the original edge. We end up with a tree T. Let T' be the multigraph obtained by doubling each edge of T. Intuitively, D_ε forces the curves P and Q to roughly trace a planar Eulerian circuit of T' using the combinatorial embedding σ. (Up to a reflection and

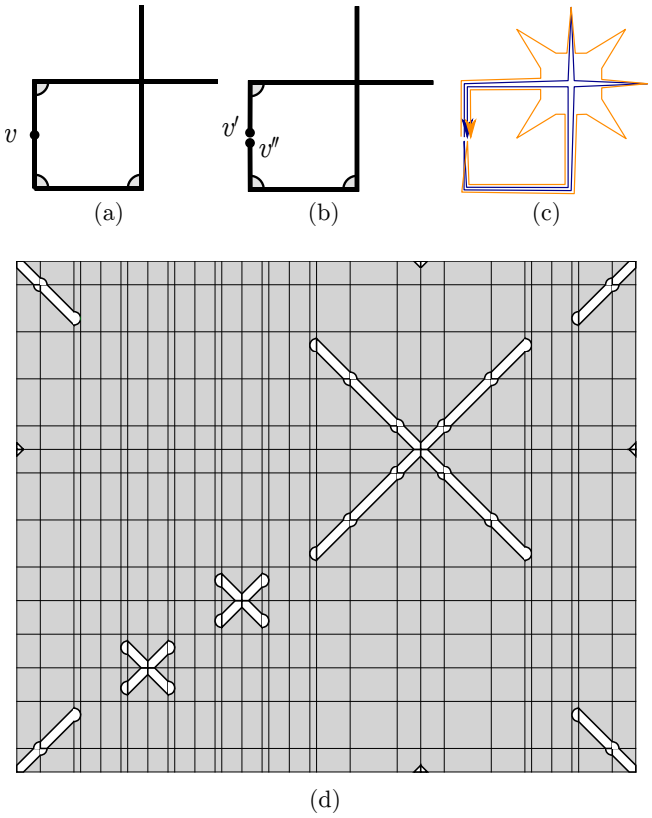


Figure 2 Example of our reduction from linkage realizability to free space diagram realizability. (a) An input linkage $\mathcal{L} = (G, \Pi)$ and a subdivision vertex v in a cycle of G. Rigid vertices are marked with gray angles. (b) Splitting v transforms G into a tree T. (c) The curves P and Q in $\mathbb{R}^2$ obtained from T. (d) The obtained free space diagram.

translation since D_ε can only specify the relative placement of P and Q.) Q is exactly a planar Eulerian circuit of T' while P traces the same circuit but avoids an ε-neighborhood of each nonrigid vertex using our *angle gadget* (described later), which allows these angles to lie freely between 60° and 240°. Both P and Q trace the "outline" σ *counterclockwise*. W.l.o.g. assume $\phi \geq 6$, scaling the linkage by a constant factor if necessary. We chose $\varepsilon = 1$ so that edges of P and Q that correspond to an edge e of G are close to each other and far from other edges. Since every partially full cell determines the relative position of the corresponding pair of edges, the four edges (two from P and two from Q) that correspond to the traversal of e are fixed relative to one another and lie on top of each other. They then simulate edge e. The angle gadget guarantees flexibility so that the angle between incident edges can vary accordingly. We add free space components to make the newly introduced subdivision vertices rigid: Their relative position is locked by Lemma 1 forming a 180° angle, see the four small components on the sides of the free space diagram in Figure 2d.

The *angle gadget*, see Figure 3, is represented by the 12 free space cells shown in Figure 3(b). It is located at a small neighborhood of a vertex v of Q; the figure only shows the portion of the free space relative to this neighborhood. Note that v is a degree-2 copy of a vertex v^* of G. For clarity, we refer to all the copies of v^* in Q with different labels (by construction, there are $\deg(v^*)$ copies of each $v^* \in V(G)$, except for the starting vertex of the Eulerian circuit, which has an extra copy). Let $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$ be the two edges of Q incident to v, and

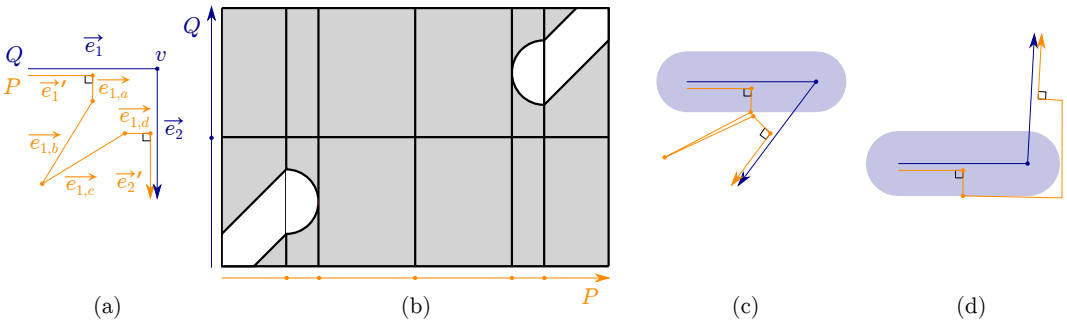


Figure 3 The angle gadget. (a) The 90° configuration and (b) its free space diagram. (c) and (d) show the extremal configurations of the gadget with angles $2 \cdot \tan^{-1}(1/2) \approx 53.13°$ and 270°, resp.

let $\overleftarrow{e_1}$ and $\overleftarrow{e_2}$ be the corresponding copies going in the opposite direction in Q, respectively. Locally, P has two edges $\overrightarrow{e_1}'$ and $\overrightarrow{e_2}'$ that overlap with $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$, respectively. We enforce the overlap by making all free space cells relative to $\overrightarrow{e_1}'$ (resp., $\overrightarrow{e_2}'$) empty except for the ones relative to $\overrightarrow{e_1}$ and $\overleftarrow{e_1}$ (resp., $\overrightarrow{e_2}$ and $\overleftarrow{e_2}$) which are partially full, containing an upward and downward 45° full strip. The distance between v and the endpoints of $\overrightarrow{e_1}'$ and $\overrightarrow{e_2}'$ closest to v is 2 by Lemma 1. We place four edges $(\overrightarrow{e_{1,a}}, \overrightarrow{e_{1,b}}, \overrightarrow{e_{1,c}}, \overrightarrow{e_{1,d}})$ between $\overrightarrow{e_1}'$ and $\overrightarrow{e_2}'$ of lengths 1, 3, 3, and 1 in this order. Only edges of length 1 have corresponding partially full cells: $C_{\overrightarrow{e_{1,a}},\overrightarrow{e_1}}$ and $C_{\overrightarrow{e_{1,a}},\overleftarrow{e_1}}$ contain half of a disk of radius 1.

▶ **Lemma 3.** *Given a realization of P and Q, assume that $(\overrightarrow{e_{1,a}}, \overrightarrow{e_{1,b}}, \overrightarrow{e_{1,c}}, \overrightarrow{e_{1,d}})$ lie to the right of $(\overrightarrow{e_1}, \overrightarrow{e_2})$. Then, $\overrightarrow{e_1}$ and $\overleftarrow{e_1}$ (resp., $\overrightarrow{e_2}$ and $\overleftarrow{e_2}$) lie exactly on top of each other, and the angle to the right of $(\overrightarrow{e_1}, \overrightarrow{e_2})$ is strictly between $2 \cdot \tan^{-1}(1/2) \approx 53.13°$ and 270°.*

Proof. The fact that $\overrightarrow{e_1}$ and $\overleftarrow{e_1}$ lie exactly on top of each other is a consequence of applying Lemma 1 to $\overrightarrow{e_1}$ and $\overrightarrow{e_1}'$, and to $\overrightarrow{e_1}'$ and $\overleftarrow{e_1}$. We now focus on the angle constraint. Note that by Lemma 1, the relative positions of $\overrightarrow{e_1}$ and $\overrightarrow{e_{1,a}}$ (resp., $\overrightarrow{e_2}$ and $\overrightarrow{e_{1,d}}$) is fixed. If we fix the positions of $\overrightarrow{e_{1,a}}$ and $\overrightarrow{e_{1,d}}$, then the positions of $\overrightarrow{e_{1,b}}$ and $\overrightarrow{e_{1,c}}$ are completely determined: There are two points whose distance is 3 from the endpoints of $\overrightarrow{e_{1,a}}$ and $\overrightarrow{e_{1,d}}$; one of them causes $\overrightarrow{e_{1,b}}$ and $\overrightarrow{e_{1,c}}$ to intersect with Q which cannot happen since their free space cells are empty. If the angle is $2 \cdot \tan^{-1}(1/2)$ or smaller, the common endpoint of $\overrightarrow{e_{1,c}}$ and $\overrightarrow{e_{1,d}}$ would lie in the closed ε-neighborhood of $\overrightarrow{e_1}$ and $C_{\overrightarrow{e_1},\overrightarrow{e_{1,c}}}$ would not be empty (Figure 3(c)), a contradiction. If the angle is 270° or greater, a portion of $\overrightarrow{e_{1,b}}$ would lie in the closed ε-neighborhood of $\overrightarrow{e_1}$ and $C_{\overrightarrow{e_1},\overrightarrow{e_{1,b}}}$ would not be empty (Figure 3(d)), a contradiction. For all values in between there is a placement for $\overrightarrow{e_{1,b}}$ and $\overrightarrow{e_{1,c}}$ away from $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$, making the section of the free space diagram exactly as required. ◀

Using Lemma 3 we can simulate a linkage $\mathcal{L}$ subject to the constraints in Theorem 2 using curves given by D_ε, obtaining the following theorem.

▶ **Lemma 4.** *It is $\exists\mathbb{R}$-complete to decide if a given free space diagram is realizable in $\mathbb{R}^2$.*

Proof. The described reduction produces a free space diagram D_ε with of size $O(|E(G)|^2)$: Q has length $2|E(G)|$ and each edge in $|E(G)|$ generates up to 10 segments in P, depending on whether the endpoints are rigid or not. The runtime is linear in the size of D_ε: each row corresponding to an edge of P has precisely two partially full cells. All other cells are empty.

Given a positive instance of linkage realization, Theorem 2(4) and Lemma 3 guarantee that we can find a placement of P and Q realizing D_ε as described in the reduction. The other direction is a little more subtle. D_ε forces Q to trace σ exactly: using Lemma 1 with

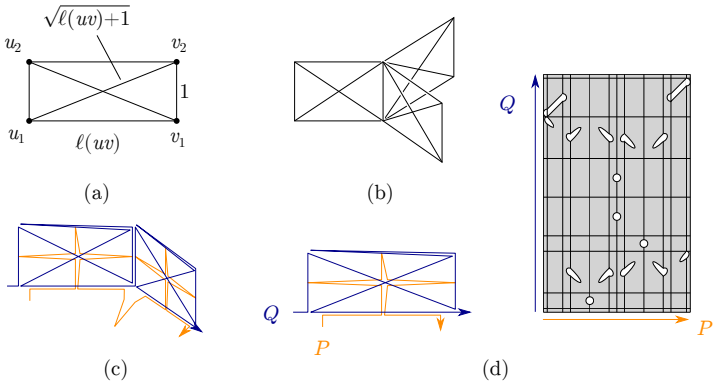


Figure 4 (a) The dimension gadget corresponding to an edge uv with length $\ell(uv)$. (b) Three gadgets in $\mathbb{R}^3$ corresponding to a degree-3 vertex. The gadget forces all vertices to be in one of two planes. (c) The realization of an angle gadget after applying the dimension gadget. (d) The free space and its realization (perturbed for clarity).

transitivity constraints the two edges of Q corresponding to an edge in $E(G)$ to lie exactly on top of each other, while the angle gadgets force the circular order around each vertex. By Lemmas 3 and 18, Q traces a noncrossing configuration of $\mathcal{L}$ exactly. If there is a valid placement of P and Q one can find a noncrossing configuration of $\mathcal{L}$ obtained by the image of Q. If such a configuration does not satisfy Theorem 2(4), that would contradict Theorem 2. Thus the promise in Theorem 2(4) must also be fulfilled by the Fréchet realization instance and the angles in each angle gadget would indeed be between 60° and 240°. ◀

3.1 Higher Dimensions

In order to show that free space realization is $\exists\mathbb{R}$-hard in higher dimensions, we show that the realizability of linkages and, thus, distance geometry are also $\exists\mathbb{R}$-hard. Here, the linkage realization is not required to be injective since we are in $\mathbb{R}^{>2}$, but the reduction will force crossings to only happen between predictable pairs of edges. This will be important in our reduction to free space realization since crossings between the curves appear in the free space diagram. We remark that, although all the ingredients of this proof were already known, the claim does not appear in the literature to the best of the authors' knowledge.

▶ **Theorem 5.** *Linkage Realization and Distance Geometry are $\exists\mathbb{R}$-hard in $\mathbb{R}^{\geq 2}$.*

Proof. Recall that linkage realization with no pinned vertices is equivalent to distance geometry. The main ingredient of this proof is the *dimension gadget* shown in Figure 4 that appears in [33]. The gadget is isomorphic to K_4 which is *globally rigid* in $\mathbb{R}^2$ [20], meaning that there is a unique embedding of the gadget in $\mathbb{R}^2$, and every realization of the gadget in $\mathbb{R}^{>2}$ is congruent with the planar realization. Given a linkage $\mathcal{L}$ satisfying the constraints of Theorem 2, replace every edge of G by a copy of the dimension gadget. Note that every vertex v is now represented by two vertices v_1 and v_2. We call the resulting linkage $\mathcal{L}'$. The gadgets force all vertices v_1 for all $v \in V(G)$ to be in the same $(k-1)$-hyperplane, perpendicular to the edges v_1v_2. Thus, $\mathcal{L}'$ is realizable in $\mathbb{R}^d$ iff $\mathcal{L}$ is realizable in $\mathbb{R}^{(d-1)}$. ◀

▶ **Theorem 6.** *It is $\exists\mathbb{R}$-complete to decide if a given free space diagram is realizable in $\mathbb{R}^{\geq 2}$.*

Proof. We adapt the dimension gadget to the free space diagram realizability problem. We first argue for $\mathbb{R}^3$. Figure 4(d) shows the adapted gadget and its free space. We use the same reduction as in Lemma 4, but replacing the edges of Q and the edges of P that overlap edges

of Q with the modified dimension gadget as follows. Each edge of Q is replaced by a path of length 7, and each edge of P that lies in the interior of an edge of Q is replaced by a path of length 10. The free space diagram between the two paths force the path of Q to be embedded as the dimension gadget. Two-dimension gadgets are neighbors if their corresponding edges share an endpoint. The cells between these paths and other non-neighbor dimension gadgets are empty. The two length-ε edges of P in the angle gadget ($\overrightarrow{e_{1,a}}$ and $\overrightarrow{e_{1,d}}$) induce partially full cells in the free space of dimension gadget (first and last collumns in the free space diagram of Figure 4(d)), forcing these edges to be perpendicular to the plane of the dimension gadget. (See Figure 4(c)) The cells of the two length-3ε edges of P in the angle gadget ($\overrightarrow{e_{1,b}}$ and $\overrightarrow{e_{1,c}}$) remain empty and thus they must be realized far from the dimension gadget. Note that the ε-neighborhood of v_1 and v_2 does not intersect P, and that the ε-neighborhood of the edges of P in the angle gadget still does not intersect Q leaving the dihedral angle between the planes of the neighbor dimension gadgets to vary as in Lemma 3. Thus the embedding of P and Q corresponds to a realization of $\mathcal{L}'$.

For dimension $d > 3$, we recursively apply the dimension gadget construction in the following way. Note that in the $d-1$-dimensional construction each edge of Q overlaps with at least one edge of P (possibly degenerate to a vertex). We recursively replace each edge of Q with the dimension gadget construction as normal. We split one edge of P that overlaps with the edge of Q at its midpoint and insert the length-8 path forming a cross that contains the midpoints of the edges in the dimension gadget of Q as in Figure 4(d). ◀

4 NP-Hardness and Algorithmic Results for Continuous Curves in $\mathbb{R}^1$

We briefly consider the realizability problem for curves in $\mathbb{R}^1$. In this case, the curves have less space to be placed in and hence the free space diagram has limited "configurations". Cells are still empty, full or partially full cells, but now free space ellipses degenerate to slabs, and the white space is bounded by parallel line segments oriented at $+$ or $-45°$, see [15, 31]. Here, we present only sketches. For details, we refer to [3]. We note that for curves in 1D the problem is weakly NP-hard by a reduction from the Partition problem. The reduction is similar to the hardness of ruler-folding.

▶ **Theorem 7.** *Realizability of continuous curves in $\mathbb{R}^1$ is weakly NP-complete.*

Next, we sketch an FPT-algorithm for continuous curves in $\mathbb{R}^1$. Inspired by computational origami [22], we observe that a given diagram D_ε is realizable iff it can be folded at the grid lines so that the white space is aligned (overlapping only with other white space) into a single convex component. In [3], we developed an algorithm to enumerate and check the different foldings, inspired by the algorithm for *simple-foldability in* $\mathbb{R}^1$ [7]. Our algorithm runs in exponential time $O(mn2^k)$, where k is the total number of (vertical and horizontal) grid lines of D_ε that do not intersect the white space (completely gray or completely white).

▶ **Theorem 8.** *Given an $m \times n$ diagram D_ε, in $O(mn2^k)$ time one can find curves P and Q in $\mathbb{R}^1$, if they exist, such that $D_\varepsilon = D_\varepsilon(P, Q)$.*

We now describe an algorithm whose input is a diagram D_ε where the dimensions of each cell are integers upper-bounded by W, and that outputs a pair of curves P, Q in $\mathbb{R}^1$ such that $D_\varepsilon = D_\varepsilon(P, Q)$ if they exist. Otherwise, it returns `false`. We use the limited placement options of curves in $\mathbb{R}^1$; the main technical ingredient is the use of dynamic programming to decide the placement of the portions of P and Q for which we have no explicit information.

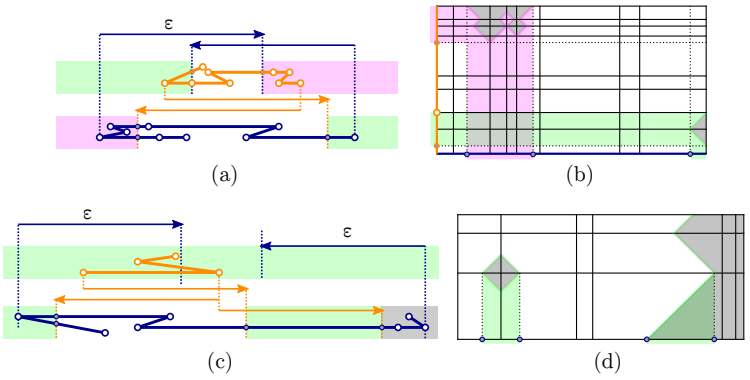


Figure 5 Regions defined by curves P (orange) and Q (blue). Certainty regions are green or pink, subdivision vertices are gray, and uncertainty regions are white (middle) or gray (left/right).

We use the following placement graph[1] G: The vertices $V(G)$ are the set of segments in the bipartite graph between segments of P and Q where an edge (v_i^P, v_j^Q) encodes that the cell $C_{i,j}$ is partially full. G can be computed in $\mathcal{O}(mn)$ time. By Lemma 3.1 in [3], if G has a single component, we can either compute P and Q, or report that no such curves exist in $\mathbb{R}^1$ in $\mathcal{O}(mn)$ time. We now show a key property of G for curves in $\mathbb{R}^1$. We say that a curve in $\mathbb{R}^1$ *spans* a distance w if its image in $\mathbb{R}^1$ is an interval of length w. A component of G is a *singleton* component if its size is one (i.e., a single vertex).

▶ **Lemma 9.** *If P and Q are two curves in $\mathbb{R}^1$, then the placement graph G computed from $D_\varepsilon(P,Q)$ has at most two non-singleton components. If either P or Q spans more than 2ε, then G has at most one non-singleton component.*

Proof. Let p_ℓ and p_r be the leftmost and rightmost points of P, respectively. We first prove the claim when P, without loss of generality, spans more than 2ε. For contradiction assume there are 2 non-singleton components in G. Every point of Q in $[p_\ell - \varepsilon, p_r + \varepsilon]$ has exactly distance ε to some point in P and thus defines the boundary of a free-space component and can be assigned an edge of G. By continuity, every maximal subcurve of Q in $[p_\ell - \varepsilon, p_r + \varepsilon]$ corresponds to edges in the same component of G. By transitivity, any two overlapping subcurves of Q in $[p_\ell - \varepsilon, p_r + \varepsilon]$ are also represented in the same component of G as both have at least one point at distance exactly ε from the same point in P. Thus the two components in G correspond to nonoverlapping maximal subcurves of Q in $[p_\ell - \varepsilon, p_r + \varepsilon]$ and there is no third subcurve of Q that overlaps the first two. Then, Q is disconnected, a contradiction.

Now, consider the case that both P and Q span less than 2ε. The points in $\mathbb{R}^1$ that are exactly at distance ε from some point in P form the intervals $[p_\ell - \varepsilon, p_r - \varepsilon]$ and $[p_\ell + \varepsilon, p_r + \varepsilon]$. The same argument as above shows that the subcurves of Q in each of these intervals define a single component in G. Thus, there are at most 2 non-singleton components. ◀

Algorithm description. First, we subdivide the two curves based on the orthogonal projections of the free space's boundary (see Figure 5). We introduce subdivision vertices so that each point in the interior of a segment is either:

1. farther than ε from any point in the other curve (an edge whose corresponding row or column in D_ε is completely empty);

[1] This is a variation of the placement graph used for the same problem for continuous curves in $\mathbb{R}^2$ [32].

2. within ε distance from every point in the other curve (an edge whose corresponding row or column in D_ε is completely full); or
3. at exactly ε distance from a point of the other curve (an edge covered by the orthogonal projection of the boundary of the free space).

We partition the segments of both curves based on these three types. Singleton components of G correspond to segments of types (1) and (2), and vertices of non-singleton components correspond to segments of type (3). The first correspond to segments of one curve that are farther than ε from every point in the other curve. The presence of type (2) segments implies that one of the curves spans less than 2ε. Adding subdivision vertices does not asymptotically increase the complexity of the problem as each segment is subdivided at most twice.

For each of the two curves, we partition $\mathbb{R}$ into up to 5 regions used to embed the segments of each of the types based on containment in the ε-neighborhoods of the extreme points of the other curve. Let p_ℓ and p_r be the leftmost and rightmost points of P. We note that we do not have previous knowledge of the points p_ℓ and p_r but we later describe how to infer information about these points from D_ε. The regions serve as an abstraction that allows us to divide the problem into subproblems. If the balls $\mathcal{B}_\varepsilon(p_\ell)$ and $\mathcal{B}_\varepsilon(p_r)$ intersect, we call the interval $[p_r - \varepsilon, p_\ell + \varepsilon]$ the *middle uncertainty region* (colored white in Figure 5(a) and (c)). Segments of type (2) must be embedded in this region. The intervals of $\mathbb{R}^1$ contained in a single disk are called *left* and *right certainty regions*, respectively $[p_\ell - \varepsilon, p_r - \varepsilon]$ and $[p_\ell + \varepsilon, p_r + \varepsilon]$ (colored green or pink in Figure 5(a) and in the bottom curve in (c)). If $\mathcal{B}_\varepsilon(p_\ell)$ and $\mathcal{B}_\varepsilon(p_r)$ do not intersect, then we call the interval $[p_\ell - \varepsilon, p_r + \varepsilon]$ the *middle certainty region* (colored green in the top curve of Figure 5(c)). The segments of type (3) must be embedded in these regions. In both cases, we call the intervals $(-\infty, p_\ell - \varepsilon]$ and $[p_r + \varepsilon, \infty)$ the *left* and *right uncertainty regions* (colored gray in Figure 5. Note that the figure only shows a closeup view and the only visible portion of a right uncertainty region is shown for the bottom curve in (c)). The segments of type (1) must be embedded in these regions.

▶ **Lemma 10.** *Given D_ε, in $O(nm)$ time we can partition the segments of Q into the three types and assign each segment to a region.*

Proof. The orthogonal projection of the components can be computed by a traversal of the free space's boundary. Thus, we can compute the subdivision vertices in $O(nm)$ time. The types of all segments can be inferred from D_ε in $O(nm)$ time. We can compute G in $O(nm)$ time. If there are two non-singleton components, we arbitrarily fix the orientation of one segment and use Lemma 3.1 in [3] to decide the relative placement for their respective segments. Note that the type of the segments adjacent to a segment in a certainty region determines which of the components is the left and which is the right certainty region: the left certainty region is adjacent to segments of type (2) on its right boundary. ◀

We can use Lemma 3.1 in [3] to determine the relative embedding of P and Q in certainty regions. It remains to determine whether the subcurves in uncertainty regions can be embedded. We use a dynamic program (DP) to solve the problem in each uncertainty region separately. We further divide the problem into two cases depending on whether we know the relative position of the boundaries of the respective region. The input of each DP is a maximal subcurve of P (resp., Q) in an uncertainty region. The DP computes the possible placements of the subcurve for a set of boundary constraints. We later describe how to combine the output of all the DPs into a single solution.

Fixed boundary subproblem. If one of the curves does not have edges of type (2), by Lemma 3.1 in [3] we know the size of the uncertainty regions. We define DP problems for each maximal subcurve in an uncertainty region whose value is `true` iff it is possible to

realize the subcurve in the region. We define subproblems based on a suffix of the subcurve and the coordinate of the first point of the suffix (details are left to a full version of this paper). The recursive definition tries embedding the next edge oriented towards the right or left, thus each subproblem depends on only two subproblems. The number of subproblems depends on the number of segments in the subcurve and the size of the uncertainty region.

▶ **Lemma 11.** *One can compute each fixed boundary subproblem defined by a subcurve Q' with n' segments, each with an integer length of at most W, and an integer interval $[0, r]$, in $O(n' \cdot \min(r, n'W))$ time.*

Proof. Since $k \in \{1, \ldots, n'\}$ and $s \in \{0, \ldots, r\}$ there are at most $O(n'r)$ subproblems. We can also upper-bound s by $n'W$ since this is the maximum length of the image of Q' (which is necessary in the case when $r = \infty$). Each subproblem can be computed in $O(1)$ time. Thus the total runtime is $O(n' \cdot \min(r, n'W))$. ◀

Variable boundary subproblem. When both curves have segments of type (2), the size of the middle uncertainty region of one curve depends on the size of the middle uncertainty region of the other. We similarly define a DP problem for each maximal subcurve in an uncertainty region. However, each subproblem is also defined by a suffix of the subcurve, the coordinate of the first point, and, additionally, the size of the uncertainty region. In this case, the size of the uncertainty region is upper-bounded by 2ε.

▶ **Lemma 12.** *For variable boundary subproblems, in $O(\max(n, m) \cdot \varepsilon^2)$ time, one can compute all DP tables and, if there exist P and Q in $\mathbb{R}^1$ that realize D_ε, find r_P and r_Q that are compatible with a solution to all subproblems, where r_P and r_Q denote the sizes of the middle uncertainty regions of P and Q respectively.*

Proof. There are $O(m + n)$ DP tables since this is the upper bound on the number of maximal subcurves in the middle uncertainty regions. Each table has $O(n' \cdot \varepsilon^2)$ subproblems, each can be computed in $O(1)$ time, where n' is the size of the maximal subcurve. Thus, it takes $O(\max(n, m) \cdot \varepsilon^2)$ to compute all DP tables. For each table, we can keep track in a separate data structure what values of α have an entry $R(i, ., \alpha) = \texttt{true}$. Then, given values for r_P and r_Q, we can check whether there exist a compatible solution in each table in $O(1)$ time per table. Thus, we can try all possible $r_P, r_Q \in \{1, \ldots, 2\varepsilon\}$ searching for values compatible with a solution for each DP problem. Then, searching for a set of compatible solutions takes $O(\max(n, m) \cdot \varepsilon^2)$ time. ◀

▶ **Theorem 13.** *Given an $m \times n$ free space diagram D_ε, where $n \geq m$, every cell has integer dimensions of at most W, and ε is an integer, we can produce two curves in $\mathbb{R}^1$ that realize D_ε or answer `false` if no such curves exist in time $O(\max(n\varepsilon^2, n^2W))$.*

5 ∃ℝ-Completeness for Discrete Curves in $\mathbb{R}^2$

We now turn to the discrete Fréchet distance, and prove that realizability by curves in $\mathbb{R}^{\geq 2}$ for a given free space matrix is also ∃ℝ-complete. We reduce from d-Stretchability, which asks whether there exists an arrangement of hyperplanes in $\mathbb{R}^{\geq 2}$ that realizes a combinatorial description. The formal description follows in the next paragraph. We use the machinery developed by Kang and Müller [29] to show that recognizing a d-sphere graph (generalization of unit disk graphs in $\mathbb{R}^d$) is ∃ℝ-hard for $d \geq 2$. (Although only NP-hardness is claimed in [29], their proof also extends to ∃ℝ-hardness as noted in their conclusion.) Recall that we explain in Section 1 that, though they are similar, our problem differs from d-sphericity.

An instance of d-Stretchability is given by a set $S \subseteq \{-,+\}^n$ of size $1+\binom{n+1}{2}$. An arrangement of n hyperplanes divides $\mathbb{R}^d$ into $1+\binom{n+1}{2}$ cells. Each vector in S corresponds to a cell in a potential arrangement. We denote by $\mathbf{v}_j \in S$ the jth vector in S and by $\mathbf{v}_j[i]$ its ith coordinate. Then $\mathbf{v}_j[i] = -$ (resp., $\mathbf{v}_j[i] = +$) if the corresponding cell is below (resp., above) the ith hyperplane. Note that $(-,\dots,-)$ and $(+,\dots,+)$ must be in S, and so we assume they are respectively $\mathbf{v}_1$ and $\mathbf{v}_2$. The problem asks whether S is the combinatorial description of an arrangement of n hyperplanes.

Reduction. Given an instance S of d-Stretchability of n hyperplanes we construct a $2n \times |S|$ free space matrix M_ε as follows. We partition P (whose vertices correspond to rows of M_ε) into two subcurves $P^+ = (a_1,\dots,a_n)$ and $P^- = (b_1,\dots,b_n)$. Informally, a_i (resp., b_i) will be a point in the upper (resp., lower) halfspace of a hyperplane ℓ_i in the arrangement. Each column j of M_ε (i.e., vertex of Q) represents a vector in $\mathbf{v}_j \in S$, that is, $M_\varepsilon[i][j] = 0$ and $M_\varepsilon[n+i][j] = 1$ (resp., $M_\varepsilon[i][j] = 1$ and $M_\varepsilon[n+i][j] = 0$) if $\mathbf{v}_j[i] = -$ (resp., $\mathbf{v}_j[i] = +$).

▶ **Theorem 14.** *Given a free space matrix M_ε, it is $\exists\mathbb{R}$-complete to decide whether there exists a pair of curves P and Q in $\mathbb{R}^{\geq 2}$ that realizes M_ε.*

Proof. Containment in $\exists\mathbb{R}$ can be proven by a straightforward reduction to $\exists\mathbb{R}$ similar to the proof of Lemma 17. We now focus on the reduction defined above. It is clear that it runs in polynomial time. Assume that there exists a pair of curves P and Q that realizes M_ε. Refer to Figure 6(a). We use the labels of point of P defined in the reduction and assume $Q = (q_1,\dots,q_{|S|})$. Recall that, informally, points q_1 and q_2 represent vectors $\mathbf{v}_1 = (-,\dots,-)$ and $\mathbf{v}_2 = (+,\dots,+)$, respectively. Rotate the solution in order to make the vector $\overrightarrow{q_1q_2}$ vertical and pointing upwards. We build a hyperplane arrangement as follows. For each $i \in [n]$, create a hyperplane ℓ_i bisecting the segment a_ib_i. Now, we argue that q_j, $j \in \{1,\dots,|S|\}$ is in a cell in the produced arrangement with description $\mathbf{v}_j$. Let C_1 and C_2 be the cells in the arrangements of circles of radius ε containing q_1 and q_2, respectively. By definition, if $\mathbf{v}_j[i] = +$, then q_j must be within ε distance from a_i and farther than ε from b_i, that is $q_j \in \mathcal{B}_\varepsilon(a_i) \setminus \mathcal{B}_\varepsilon(b_i)$. Thus, $C_1 = (\bigcap_{i=1}^n \mathcal{B}_\varepsilon(b_i) \setminus \bigcup_{i=1}^n \mathcal{B}_\varepsilon(a_i))$ and $C_2 = (\bigcap_{i=1}^n \mathcal{B}_\varepsilon(a_i) \setminus \bigcup_{i=1}^n \mathcal{B}_\varepsilon(b_i))$. Note that every hyperplane ℓ_i must separate C_1 and C_2 by definition. Thus every ℓ_i intersects the line segment $\overline{q_1q_2}$. We focus on a specific hyperplane ℓ_i. Without loss of generality assume $\mathbf{v}_j[i] = +$. Then, $\mathcal{B}_\varepsilon(a_i) \setminus \mathcal{B}_\varepsilon(b_i)$ is above ℓ_i and so is q_j. Therefore, the produced hyperplane arrangement realizes S.

Now assume that there exists a hyperplane arrangement realizing S. Refer to Figure 6(b). For each cell in the arrangement described by $\mathbf{v}_j$, choose a point q_j in the interior of the cell. As before, every hyperplane intersects the line segment $\overline{q_1q_2}$, since q_1 is below all the hyperplanes and q_2 is above. Let t_i be the intersection of ℓ_i and $\overline{q_1q_2}$. Define the balls $\mathcal{B}_r(w^+_{i,r})$ and $\mathcal{B}_r(w^-_{i,r})$ respectively above and below ℓ_i, tangent to ℓ_i at t_i. Note that $\mathcal{B}_r(w^+_{i,r})$ (resp., $\mathcal{B}_r(w^-_{i,r})$) equals the upper (resp., lower) halfspace of ℓ_i when $r \to \infty$. Thus, for sufficiently large r, $\mathcal{B}_r(w^+_{i,r})$ contains all points q_j above ℓ_i and $\mathcal{B}_r(w^-_{i,r})$ contains all points q_j below ℓ_i. Let r^* be a sufficiently large r such that the previous statement is true for all $i \in [n]$. Scale the entire construction to make $r^* = \varepsilon$. Then, we can construct P by making $a_i = w^+_{i,\varepsilon}$ and $b_i = w^-_{i,\varepsilon}$. Now, each q_j is contained in the appropriate cell of the arrangement of circles of radius ε centered at points of P. Thus, the constructed P and Q realize M_ε. ◀

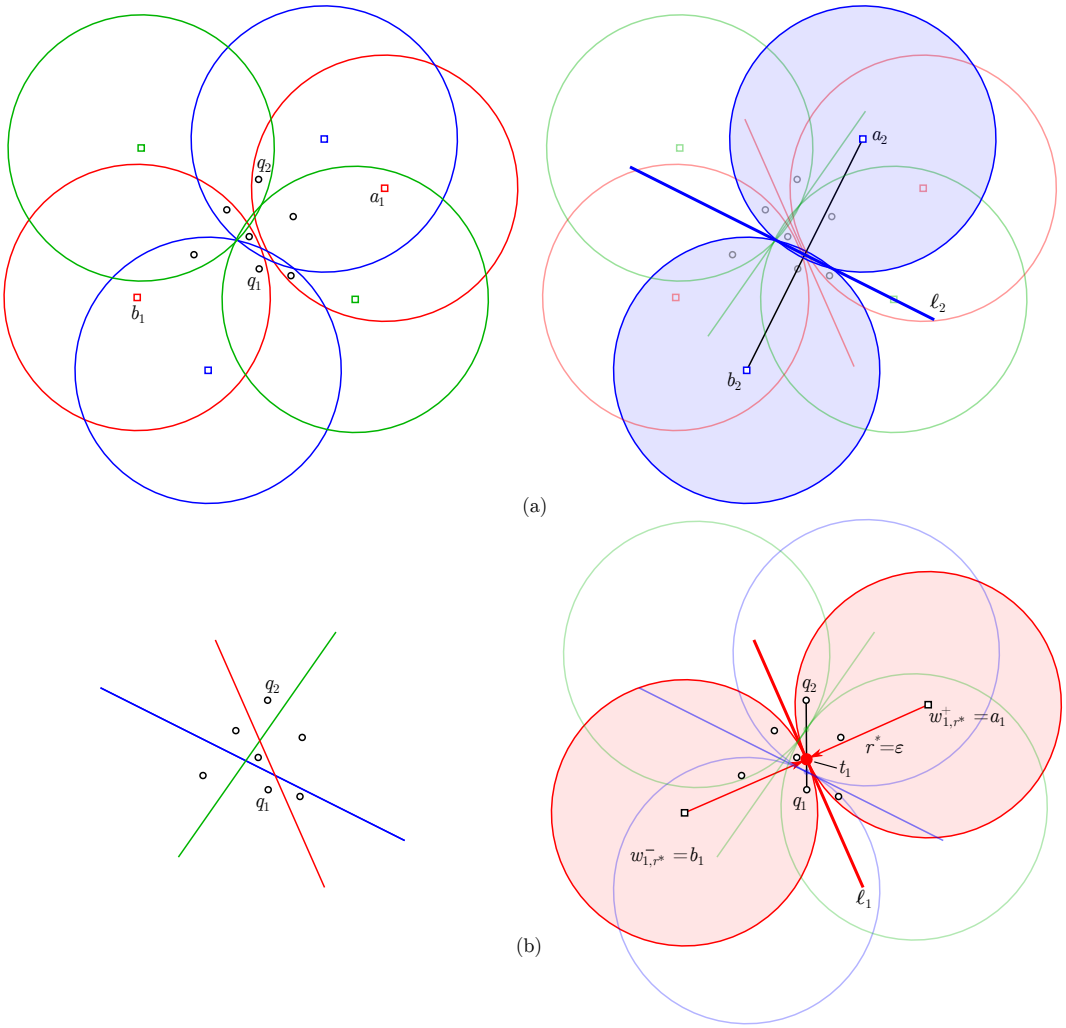


Figure 6 Reduction from $S = \{(-,-,-),(+,+,+),(-,-,+),(-,+,+),(-,+,-),(+,+-),(+,-,-)\}$. (a) Transforming a solution to M_ε into a line arrangement that realizes S. (b) Transforming a solution to S into curves P and Q that realize M_ε. Here squares represent points of P and circles represent points of Q.

6 A Polynomial Time Algorithm for Discrete Curves in $\mathbb{R}^1$

We now turn to realizability for discrete curves in $\mathbb{R}^1$ and show that this can be decided in polynomial time. We lend our main idea from the *unit-interval graph recognition* (UIGR) in [21]: given an abstract graph G whose nodes are intervals and two intervals intersect iff there is an edge between the two corresponding nodes in G, the goal is to find a placement of intervals in the real line such that the intersections induced by them fulfill G. See Figure 7 for an example. In free space realizability, we derive a unit interval graph G from the free space matrix M_ε. We then adapt the idea in [21] to handle the realizability in our case.

First, we borrow some notations from [21]. For a vertex $v \in G$ the neighboring vertices of v is denoted by $N(v)$. We also define $N[v] = N(v) \cup \{v\}$. Two vertices u and v are *indistinguishable* if $N[u] = N[v]$. In order to recognize a unit-interval graph, Corneil et al. [21] propose a linear-time algorithm: (i) find the left anchor in G (the left-most interval in

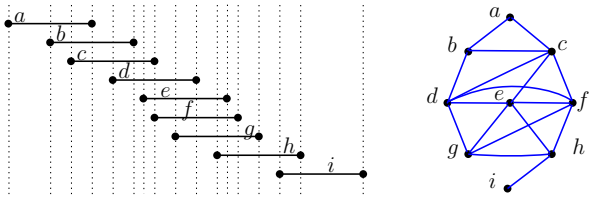


Figure 7 An abstract graph of intervals and its recognition in $\mathbb{R}^1$.

the recognition), (2) perform a BFS search starting at the left anchor to get a *partial order* of the intervals, meaning that some groups of intervals are ordered properly, but still intervals belonging to each group need to be re-ordered, and (3) refine the partial order, i.e., the intervals within in each group, to get the *global order*. If the global order exists, return "YES", and "NO", otherwise. To handle (1), they perform a BFS from an arbitrary node in G. Find a vertex z at the last level L_t of the BFS search such that $\deg(z) = \min\{\deg(w) : w \in L_t\}$. In (2), they locally order the vertices in G based upon the level in which they are encountered along the BFS search from z. Finally in (3), in each level L_k obtained in (2) they sort each vertex $v \in L_k$ in an increasing order of $D(v) = |\mathsf{Next}(N(v))| - |\mathsf{Prev}(N(v))|$. Here, $\mathsf{Next}(v)$ is the set of adjacent vertices to v in L_{k+1} and $\mathsf{Prev}(v)$ is the set of adjacent vertices to v in L_{k-1}. Next, for each vertex v in G, they compute $\alpha(v) = \min\{\text{order}(u) : \ u \in N[v]\}$ and $\omega(v) = \max\{\text{order}(u) : \ u \in N[v]\}$, where $\text{order}(u)$ is the order in which u is encountered on the line. In the end, if there exists a v for which $\alpha(v) \neq \omega(v)$, "NO" is returned, and "YES" is returned, otherwise. We modify Step (3) to handle our case.

The problem of realizing M_ε is more restrictive than UIGR as follows. Let $\varepsilon = 1/2$. Similar to the $\mathbb{R}^2$ case, we can see the realization of $Q = (q_1, \ldots, q_n)$ as an arrangement of intervals $\mathcal{B}_\varepsilon(q_j)$ partitioning $\mathbb{R}^1$. Each row i of M_ε describes a "cell" in the arrangement (which is an interval in $\mathbb{R}^1$) where we place a point p_i of P. Thus, M_ε requires a unit interval realization of P with not only prescribed adjacencies in G but prescribed cells. Our goal is to take the partial order that we get from (2), and refine it to be closer to a global order using information from M_ε. In the following, we refer to vertices of G and the interval they refer to interchangeably, and we call the maximal set of vertices that are pairwise indistinguishable an *equivalence class*. See Appendix B for a full algorithmic description.
DiscreteRealizabilityAlgo(M_ε) : **(1)** Construct the unit-interval graph G from M_ε. **(2)** Choose a left anchor v_0 by running a BFS search on G. **(3)** Perform a BFS on G starting at v_0 to obtain a partial order of the intervals. **(4)** Refine the partial order by sorting the intervals at each level of the BFS under the criterion of $D(v) = |\mathsf{Next}(N(v))| - |\mathsf{Prev}(N(v))|$. **(5)** Refine the partial order D to an order D': For each row of M_ε, place intervals of entry 1 in the equivalence class C towards those intervals that don't belong to C whose entries are 1 and orders are different than intervals in C. **(6)** Extend the partial order defined by the BFS levels and D' to a global order breaking ties arbitrarily. **(7)** Verify whether the produced arrangement is compatible with M_ε or not.

▶ **Lemma 15.** *The algorithm returns "YES" if and only if M_ε is realizable.*

Proof. By Step (7), it is clear that when the algorithm returns "YES" the instance M_ε is realizable. If the algorithm returns "NO", we show that there are no P and Q realizing M_ε. Note that the constraints in the ordering obtained from Steps (1–4) are the same as for UIGR. Thus, we must show that if G is not a unit interval graph, then M_ε is not realizable. We show the constrapositive: if M_ε is realizable, then G is a unit interval graph. As discussed before, the realization of Q implies the realization of a unit interval graph G^* whose intervals

are $\mathcal{B}_\varepsilon(q_j)$, for $\varepsilon = 1/2$. It is clear that G^* is a supergraph of G since the required cells in the interval arrangement exist containing points of P. Let q_iq_j be an edge that exists in G^* and not in G such that q_i is to the left of q_j and with shortest interval $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$. Since q_iq_j is not in G, $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$ contain no points of P and all the cells in this interval are not necessary. By the assumption that $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$ is shortest, there is no point q_k of Q to the right of q_j whose interval $\mathcal{B}_\varepsilon(q_k)$ intersects $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$. Let S be the set of points including q_j and all points q_k of Q to the right of q_j whose interval $\mathcal{B}_\varepsilon(q_k)$ does not intersect $\mathcal{B}_\varepsilon(q_j)$. Move all points in S until the intersection $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$ disappears. By construction, ass cells previously in $\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)$ disappear. We show that no other cell does, and thus the modified Q is still a solution for M_ε. A cell to the left of $q_j - \varepsilon$ is not affected since we only move points to the left of q_j. A cell to the right of $q_j + \varepsilon$ would disappear if $\mathcal{B}_\varepsilon(q_j)$ starts to intersect with an interval that it didn't before. This does not happen since S contains all such intervals. A cell in $\mathcal{B}_\varepsilon(q_j) \setminus \mathcal{B}_\varepsilon(q_i)$ would disappear if an interval defined by $q_k \in S \setminus \{q_j\}$ stopped intersecting another interval. Recall that there are no intervals whose right endpoint are between $q_j + \varepsilon$ and $q_j + \varepsilon + |\mathcal{B}_\varepsilon(q_i) \cap \mathcal{B}_\varepsilon(q_j)|$ by the "shortest" assumption. Then, such cells do not exist. Thus, we produced a solution to M_ε whose interval intersection graph has fewer edges than G^*. Applying this argument successively we conclude that there exist a solution whose intersection graph is G.

We now show that Step (5) is necessary. Suppose there are four intervals $\{a, b, c, d\}$ in row r, $I_r = \{a, c\}$, $C = \{b, c, d\}$, and $C' = \{c\}$. Also by assumption $a <_D b =_D c =_D d$. For the sake of contradiction, suppose that there exists a positive solution that does not place the interval c before $C \setminus C' = \{b, d\}$ and after $I_r \setminus C = \{a\}$. Placing c before a implies that $c <_D a$ which is a contradiction. Placing c after b implies that the intersection $\mathcal{B}_\varepsilon(a) \cap \mathcal{B}_\varepsilon(c)$ is contained in $\mathcal{B}_\varepsilon(b)$. This means that the cell required by $I_r = \{a, c\}$ does not exist, which is again a contradiction.

Finally, we analyse Steps (6–7). The only worry is that there might exist two extensions of the partial order produced in Step (5) such that one is a positive solution and the other isn't. Let q_i and q_j be two incomparable vertices in the partial order. Since Step (5) refines equivalence classes, q_i and q_j are also in the same equivalence class C. Then, there exist no row containing an interval not in C whose entries relative to q_i and q_j are different. If there is a row containing only a proper subset of C, the arrangement must contain a cell in which the proper subset of C intersect and that does not intersect any other interval. Since intervals in C intersect the same intervals by definition, including intervals that must be to the left and to the right of intervals in C (note that we add v_0 and v_f so that every equivalence class has a predecessor and successor), this cell cannot exist. Then Step (7) returns "NO". Else, the columns relative to q_i and q_j are identical and interchangeable. ◀

The construction of G takes $O(k^2n)$ time where $k \leq m$ is the maximum number of entries filled with 1 over all rows in M_ε, since G might contain n cliques of size k. The other steps can be implemented in linear time. The full algorithm description is given in Appendix B.

▶ **Theorem 16.** *Given an $n \times m$ free space matrix M_ε, we can decide whether there exist curves P and Q in $\mathbb{R}^1$ that realize M_ε in $O(nm + k^2n)$ time, where $m \leq n$.*

Proof. Correctness is given by Lemma 15. First note that the size of V derived from M_ε is $|V| = m$, and $|E| = O(m^2)$ due to the size of the adjacency matrix we use in Step (1), however, we need $O(k^2n)$ time to add all edges in G due to the size of the clique induced by the intervals with entry 1 in each row. The size of each clique is at most k^2. Thus $|E| = \min(m^2, k^2n)$. Steps (2–3) takes $O(|V| + |E|) = O\big(m + \min(m^2, k^2n)\big)$ for performing the BFS on G. Step (4) takes $O(|V|) = O(m)$ time using a counting sort per level of the

BFS. Step (5) takes $O(mn)$ time by processing each row of M_ε and partitioning the relevant intervals into their equivalence classes in $O(m)$ time. Step (6) takes $O(m)$ time. In step (7), the real line can be partitioned into $2m = O(m)$ cells. Verifying that all n points of P fall into the induced cell takes $O(nm)$ time. Thus, the algorithm's total runtime is $O(mn + k^2 n)$. ◀

References

1 Zachary Abel, Erik D Demaine, Martin L Demaine, Sarah Eisenstat, Jayson Lynch, and Tao B Schardl. Who needs crossings? Hardness of plane graph rigidity. In *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51, pages 3:1–3:15, 2016.

2 Zachary Ryan Abel. *On folding and unfolding with linkages and origami.* PhD thesis, Massachusetts Institute of Technology, 2016.

3 Hugo A. Akitaya, Maike Buchin, Majid Mirzanezhad, Leonie Ryvkin, and Carola Wenk. Realizability of free space diagrams for 1D curves. In *38th European Workshop on Computational Geometry (EuroCG)*, 2022.

4 Hugo Alves Akitaya, Maike Buchin, Leonie Ryvkin, and Jérôme Urhausen. The k-Fréchet distance: How to walk your dog while teleporting. In *30th International Symposium on Algorithms and Computation, ISAAC 2019*, pages 50:1–50:15, 2019.

5 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(1-2):75–91, 1995.

6 Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica. An International Journal in Computer Science*, 38(1):45–58, 2004.

7 Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Martin L. Demaine, Joseph S.B. Mitchell, Saurabh Sethia, and Steven S. Skiena. When can you fold a map? *Computational Geometry*, 29(1):23–46, 2004. Special Issue on the 10th Fall Workshop on Computational Geometry, SUNY at Stony Brook.

8 Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet distance for curves, revisited. In *14th Annual European Symposium on Algorithms*, pages 52–63, 2006.

9 Peter F Ash and Ethan D Bolker. Recognizing Dirichlet tessellations. *Geometriae Dedicata*, 19:175–206, 1985.

10 Jérémy Barbay. Adaptive computation of the discrete fréchet distance. In *String Processing and Information Retrieval*, pages 50–60, 2018.

11 Hossein Boomari, Mojtaba Ostovari, and Alireza Zarei. Recognizing visibility graphs of polygons with holes and internal-external visibility graphs of polygons. *arXiv preprint*, 2018. `arXiv:1804.05105`.

12 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670, 2014.

13 Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *Journal of Computational Geometry*, 7(2):46–76, 2016.

14 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog: Improved bounds for computing the fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017.

15 Kevin Buchin, Jinhee Chun, Maarten Löffler, Aleksandar Markovic, Wouter Meulemans, Yoshio Okamoto, and Taichi Shiitada. Folding free-space diagrams: Computing the Fréchet distance between 1-dimensional curves (multimedia contribution). In *33rd International Symposium on Computational Geometry, (SoCG 2017)*, pages 64:1–64:5, 2017.

16 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. SETH says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *Proc. 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2887–2901, 2019.

17 Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry (SOCG'14)*, pages 367–376, 2014.

18 Maike Buchin, Leonie Ryvkin, and Carola Wenk. On the realizability of free space diagrams. In *37th European Workshop on Computational Geometry (EuroCG)*, 2021.

19 Jean Cardinal and Udo Hoffmann. Recognition and complexity of point visibility graphs. *Discrete & Computational Geometry*, 57, January 2017.

20 Robert Connelly. Generic global rigidity. *Discrete & Computational Geometry*, 33(4):549, 2005.

21 Derek G Corneil, Hiryoung Kim, Sridhar Natarajan, Stephan Olariu, and Alan P Sprague. Simple linear time recognition of unit interval graphs. *Information Processing Letters*, 55(2):99–104, 1995.

22 Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, 2008.

23 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48:94–127, 2012.

24 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry. An International Journal of Mathematics and Computer Science*, 48(1):94–127, 2012.

25 Alon Efrat, Leonidas J. Guibas, Sariel Har-Peled, Joseph S.B. Mitchell, and T.M Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete & Computational Geometry*, 28(4):535–569, 2002.

26 Joachim Gudmundsson, Patrick Laube, and Thomas Wolle. Movement patterns in spatio-temporal data. In S. Shekhar and H. Xiong, editors, *Encyclopedia of GIS*. Springer-Verlag, 2007.

27 Timothy Franklin Havel. *The combinatorial distance geometry approach to the calculation of molecular conformation*. University of California, Berkeley, 1982.

28 Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure-structure alignment with discrete Fréchet distance. *Journal of Bioinformatics and Computational Biology*, 6(1):51–64, 2008.

29 Ross J Kang and Tobias Müller. Sphere and dot product representations of graphs. *Discrete Comput Geom*, 47:548–568, 2012.

30 Joseph Noggle. *The nuclear Overhauser effect*. Academic Press, 2012.

31 Günter Rote. Lexicographic Fréchet matchings. In *30th European Workshop on Computational Geometry*, 2014.

32 Leonie Ryvkin. *On distance measures for polygonal curves bridging between Hausdorff and Fréchet distance*. doctoralthesis, Ruhr-Universität Bochum, Universitätsbibliothek, 2021. doi:10.13154/294-8275.

33 James B Saxe. Embeddability of weighted graphs in k-space is strongly NP-hard. In *17th Allerton Conf. Commun. Control Comput., 1979*, pages 480–489, 1979.

34 Marcus Schaefer. Realizability of graphs and linkages. In *Thirty Essays on Geometric Graph Theory*, pages 461–482. springer, 2012.

35 R. Sriraghavendra, Karthik K., and Chiranjib Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR)*, pages 461–465, 2007.

A Omitted Proofs and Details from Section 3

▶ **Lemma 17.** *The problem of finding curves P and Q in $\mathbb{R}^2$ that realize an input diagram D_ε is in $\exists\mathbb{R}$.*

Proof. We reduce the problem to a system of real polynomial inequalities. The coordinate of vertices of P and Q are the variables. Each cell represents a constraint: The proof of Lemma 3.1 [3] gives a quadratic equation relating the endpoints of the corresponding segments.

Completely full cells require each segment to lie in the intersection of the ε-neighborhood of the other segment's endpoints. Completely empty cells require that each segment lies outside of the other segment's ε-neighborhood. All constraints can be expressed with a constant number of quadratic inequalities. ◀

Reduction description. Here, we detail the overview shown in Section 3. Refer to Figure 2. Note that every edge in Q corresponds to an edge of T. Every edge of T has two correspondents on Q. Every edge in P either corresponds to an edge of T or is part of an angle gadget. A free space cell is empty unless it corresponds to a pair of edges of P and Q that (i) correspond to the same edge, (ii) correspond to adjacent edges and their shared vertex is rigid, (iii) one is part of an angle gadget and the other corresponds to one of the edges in the angle represented by the gadget, or (iv) correspond to a pair of edges in T that were the result of a subdivision of an edge of G. In case (i), the corresponding cell has a $\pm 45°$ slab of width $\sqrt{2}$ depending on the direction of the traversal of the edge. In case (ii), if the rigid acute angle between the corresponding edges of G is $90°$, the corresponding cell has a quarter of a unit disk centered at the corner that corresponds to the rigid vertex. Else, the rigid angle is $180°$ the corresponding cell has a right isosceles triangle with two edges of length 1 and whose vertex incident to the right angle is at the corner of the cell that corresponds to the rigid vertex. In case (iii), as explained in the description of the angle gadget, the cells corresponding to long edges are empty. The cells between a short edge of P and an edge of Q incident to the angle will contain half of a unit disk so that the distance between the half-disk and the corner of the cell that correspond to the nonrigid vertex is 1. Finally, case (iv) is the same as case (ii) with a rigid angle of $180°$.

▶ **Lemma 18.** *Given a realization of P and Q, the subcurves that correspond to a rigid subgraph H exactly cover a rigid transformation of H.*

Proof. By construction, every pair of edges from P and Q that either correspond to the same edge of T, or to two adjacent edges of T that in turn correspond to edges in the same rigid subgraph H, define a partially full cell. Then, the claim follows by Lemma 1. ◀

B Omitted Details from Section 6

The full description of realizability of discrete curves in $\mathbb{R}^1$ is presented below:
DiscreteRealizabilityAlgo(M_ε) :

Step (1): Construct an abstract unit-interval graph G from M_ε whose rows are vertices in P and columns are vertices in Q: The vertex set of G represents the columns of M_ε, i.e., the interval of length 2ε centered at a point $q \in Q$. For every row ($p \in P$), the edge set of G is defined by including a clique between the columns (vertices in Q) that are filled with 1 in that row. We store the edges of G in an adjacency matrix. In the following we assume G connected, or else, we treat each component separately.

Step (2): Choose a left anchor as follows. As in [21], we get a set of candidates for left anchor by running a BFS search on G from an arbitrary vertex. The set contains the vertices with minimum degree in the deepest level of the BFS. The set of candidates must be in at most two equivalence classes, or else G is not an interval graph. We augment G by adding a new vertex v_0 connected to each vertex in one of the equivalence classes of candidates. We choose v_0 as a left anchor.

Step (3): Perform a BFS on G starting at v_0 to obtain a partial order of the vertices.

Step (4): Refine the partial order to get the global order by sorting the intervals at each level of the BFS under the criterion of $|\mathsf{Next}(N(v))| - |\mathsf{Prev}(N(v))|$. We denote the current partial order as D and use $<_D$, $>_D$ and $=_D$ to denote whether intervals appear in order, in reverse order, or are incomparable in D, respectively. By Theorem 2.2 in [21], a pair of vertices u and v with $u =_D v$ are indistinguishable. Thus a set of pairwise incomparable vertices in this partial order forms an equivalence class. Add a vertex v_f to the end of the order, connecting it to all vertices in the last equivalence class.

Step (5): Further refine the partial order as follows. We refer to such refinement as D'. For each row r in M_ε, let I_r be the set of intervals with entry 1 in r, and let $C' = C \cap I_r \neq \emptyset$ where C is an equivalence class. If there are $i \in I_r \setminus C$ and $c' \in C'$ where $i <_D c'$ (resp., $i >_D c'$), make $c' <_{D'} c$ (resp., $c' >_{D'} c$) for all $c \in C \setminus C'$.

Step (6): Extend the partial order defined by the BFS levels and D' to a global order breaking ties arbitrarily. Obtain the arrangement of unit intervals based on the global order and G. This can be done as in Theorem 3.2 in [21].

Step (7): Verify whether the produced arrangement is compatible with M_ε. Each row r specifies the existence of a cell where exactly the intervals with entry 1 intersect. If a cell specified by a row does not exist in the arrangement, return "NO". Otherwise, return "YES".

k-Universality of Regular Languages

Duncan Adamson
Leverhulme Centre for Functional Material Design, University of Liverpool, UK

Pamela Fleischmann
Department of Computer Science, Kiel University, Germany

Annika Huch
Department of Computer Science, Kiel University, Germany

Tore Koß
Department of Computer Science, University of Göttingen, Germany

Florin Manea
Department of Computer Science, University of Göttingen, Germany

Dirk Nowotka
Department of Computer Science, Kiel University, Germany

Abstract

A subsequence of a word w is a word u such that $u = w[i_1]w[i_2]\dots w[i_k]$, for some set of indices $1 \leq i_1 < i_2 < \cdots < i_k \leq |w|$. A word w is k-subsequence universal over an alphabet Σ if every word in Σ^k appears in w as a subsequence. In this paper, we study the intersection between the set of k-subsequence universal words over some alphabet Σ and regular languages over Σ. We call a regular language L k-$\exists$-*subsequence universal* if there exists a k-subsequence universal word in L, and k-$\forall$-*subsequence universal* if every word of L is k-subsequence universal. We give algorithms solving the problems of deciding if a given regular language, represented by a finite automaton recognising it, is k-$\exists$-*subsequence universal* and, respectively, if it is k-$\forall$-*subsequence universal*, for a given k. The algorithms are FPT w.r.t. the size of the input alphabet, and their run-time does not depend on k; they run in polynomial time in the number n of states of the input automaton when the size of the input alphabet is $O(\log n)$. Moreover, we show that the problem of deciding if a given regular language is k-$\exists$-*subsequence universal* is NP-complete, when the language is over a large alphabet. Further, we provide algorithms for counting the number of k-subsequence universal words (paths) accepted by a given deterministic (respectively, nondeterministic) finite automaton, and ranking an input word (path) within the set of k-subsequence universal words accepted by a given finite automaton.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases String Algorithms, Regular Languages, Finite Automata, Subsequences

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.4

Related Version *Full Version*: `http://arxiv.org/abs/2311.10658`

Funding *Duncan Adamson*: Leverhulme Trust via the Leverhulme Research Centre for Functional Material Design
Tore Koß: partly supported by the German Research Foundation (DFG), via the project number 389613931 (research grant)
Florin Manea: partly supported by the German Research Foundation (DFG), via the project number 466789228 (Heisenberg grant)

1 Introduction

Words and subsequences are two fundamental combinatorial objects. Informally, a subsequence of a word w is a word u that can be obtained by deleting some of w's letters while preserving the order of the rest. For instance, `taunt` and `salty` are sub-

sequences of `automatauniversality`, while `trauma` is not since these letters do not occur in the correct order. Subsequences are a heavily studied object within computer science [6, 9, 13, 22, 31, 35, 38, 40, 45, 49, 50, 51] and beyond, with applications in a wide number of fields including bioinformatics [23, 47], database theory [5, 17, 32, 33], and modelling concurrency [46]. A survey of combinatorial pattern matching algorithms for subsequences has been provided by Kosche et al. [36], highlighting a series of recent results for problems on finding subsequences in words as well as their applications and connections to other areas of computer science, to which we refer the reader for further details and references.

This paper considers *k-subsequence universal* words. A word w is k-subsequence universal over an alphabet $\Sigma = \{1, \ldots, \sigma\}$ if w contains every word of length k over Σ as a subsequence. These words were first defined by Karandikar and Schnoebelen [28, 44] as *k-rich words*, however more recent work has used the term *k-subsequence* (or *scattered factor*) *universality* [2, 4, 6, 9, 12, 14, 35, 45], which we use here. The study of these words follows from the seminal work by Simon [48] where a congruence – nowadays known as *Simon's congruence* – is introduced. Two words w, v are k-congruent, denoted $w \sim_k v$, if w and v share the same set of subsequences up to length k. As such, k-subsequence universal words are those which are k-congruent to the word $(1 \cdots \sigma)^k$.

Simon's congruence relation is well studied [11, 13, 14, 49, 50, 51], with recent asymptotically optimal algorithms for testing if two words are k-congruent [6] and for computing the largest k for which two words are k-congruent [18], as well as for pattern matching under Simon's congruence [31]. Indeed, besides the usage of k-subsequence universal words in [28, 44] in a context related to the study of the height of piecewise testable languages and the logic of subsequence, the idea of universality itself is quite important in formal languages, automata theory, but also in combinatorics. In this context, the universality problem [25] is whether a given language L (over an alphabet Σ, given as an automaton) is equal to Σ^*. This problem for various classes of languages and language accepting/generating formalisms is studied in, e.g., [41, 37, 19] and the references therein. The universality problem was considered for contiguous factors (substrings) of words [39, 10] and partial words [7, 21], as well. In that context, one analyses, the (partial) words w over an alphabet Σ which have exactly one occurrence of each string of length ℓ over Σ as a substring. De Bruijn sequences [10] fulfil this property and have many applications in computer science or combinatorics, see [7, 21] and the references therein. While it is a perfectly valid problem to investigate (partial) words where each substring occurs exactly once, in the case of subsequence universality this is a trivial restriction, as in each long-enough word there will be subsequences occurring more than once [6].

Coming closer to the topic of this work, we recall the works by Barker et al. [6], Day et al. [9], and Schnoebelen and Veron [45], which directly address k-subsequence universal words. In [6], the authors show that it is possible to determine in linear time (1) whether a word is k-subsequence universal and (2) the shortest k-subsequence universal prefix of a given word. Additionally, they show that the minimal set of factors $w_1, w_2, \ldots, w_\ell$ of a word w, such that $w_1 w_2 \cdots w_\ell$ is k-subsequence universal and the exponent i such that w^i is k-subsequence universal can be determined efficiently. Further, [9] proposes a set of algorithmic results for computing the minimum number of edit operations (insertion, deletions, substitutions) to apply to a word w in order to make it k-subsequence universal; in general, their algorithms run in $O(|w|k)$ time, for $k \leq |w|$ (the problems are trivial, otherwise). Interestingly, the problems approached in that paper can be seen as determining the minimum number Δ such that the (finite) language containing the words found at edit distance at most Δ from w contains a k-universal word. Finally, [45] presents an algorithm computing the largest k for which a word, given in a compressed form, is k-universal.

Another paper which is also strongly related to our work is [29] (as well as its follow-up [30]). In this paper, the authors investigate the language of words which are k-congruent to u, for a given word u, called the k-closure of u. They show that this language is regular, and effectively construct a finite automaton recognising it (of size exponential in σ, the size of the input alphabet). Now, testing whether another given finite (or regular) language contains a word which is k-congruent to u can be reduced to deciding the emptiness of the intersection between the k-closure of u and the given language.

Our work builds on [9, 29], as well as on the works whose main object is the downward closure of languages (see [51] and the references therein), and investigates the problem of efficiently detecting k-subsequence universal words within regular languages. In other words, we are interested in identifying the words of a given regular language L which are k-subsequence universal, i.e., in the k-closure of the target word $(1 \cdots \sigma)^k$. There is, however, a fundamental difference between the problems considered here and those of [29, 30]. The input of the problems studied here is a regular language L and a number k (given in binary representation, similarly to [9]), and we want to detect the words of L which are k-universal, without having to explicitly write those words or the target word $(1 \cdots \sigma)^k$ (whose length is at least $k\sigma$, so exponential in the size of the binary representation of k, our input). On the other hand, in the setting of [29, 30] we are explicitly given a target word w for which we construct the finite automaton recognising its k-closure; applying this approach to our setting would lead to dealing explicitly with the target word $w = (1 \cdots \sigma)^k$, so we would directly have an exponential blow-up both at this step and when the automaton is constructed. Hence, the problem discussed in this paper extends significantly the one of [9] by looking at the intersection between the language of k-subsequence universal words and arbitrary regular languages, rather than a very particular class of finite languages. Moreover, it addresses a highly-relevant particular case of the theory presented in [29, 30], by considering the class of k-subsequence universal words, with the interesting property that this case can be succinctly specified, and with the hope that more direct and efficient algorithms can be obtained in this setting, without having to go through the general framework.

Going more into the details of our approach, we start our investigation by defining two notions for k-subsequence universality of regular languages. A regular language L is *existence k-subsequence universal* (k-$\exists$-subsequence universal) if there exists at least one k-subsequence universal word in L, and it is *universal k-subsequence universal* (k-$\forall$-subsequence universal) if every word of L is k-subsequence universal. We assume that regular languages are given by finite automata which recognises them, so, canonically, we will call an automaton k-$\exists$-subsequence universal (respectively, k-$\forall$-subsequence universal) if the language accepted by it is k-$\exists$-subsequence universal (respectively, k-$\forall$-subsequence universal).

Alongside the categorisation problems (given an automaton, decide whether the language it recognises is k-$\exists$-subsequence universal or k-$\forall$-subsequence universal), we consider the problems of *counting* and *ranking* the number of ℓ-length k-subsequence universal words accepted by a given finite automaton $\mathcal{A}$. The counting problem asks for the total number of ℓ-length k-subsequence universal words accepted by $\mathcal{A}$. The ranking problem takes a word w as input and asks for the number of k-subsequence universal words accepted by $\mathcal{A}$ that are lexicographically smaller than w. Both of these problems have been heavily studied for other classes of words, including cyclic words [1, 2, 3, 16, 20, 34, 43] and Gray codes [16, 34, 42].

Our Contributions. In Section 2 we introduce the novel notions of k-$\exists$-subsequence universality or k-$\forall$-subsequence universality for regular languages and finite automata.

In Section 3, we give algorithms solving the problems of deciding if the language accepted

by a given finite automaton with n states is k-$\exists$-*subsequence universal* and, respectively, if it is k-$\forall$-*subsequence universal*, for a given k. These algorithms are fixed parameter tractable with respect to σ, the size of the input alphabet. If we have additionally $\sigma \in O(\log n)$, they run in polynomial time, and their run-time does not depend at all on k. Note that one could easily devise solutions for both these problems using the framework of [29, 30], but their complexity would have been exponential both in $\log k$ (the size of the representation of k in our input) and in σ. Moreover, we show that, if no bound is placed on the size of the input alphabet, the problem of deciding if a given regular language is k-$\exists$-*subsequence universal* is NP-complete. The NP-hardness of this problem follows from [29]; it is worth noting that, on the one hand, the problem is hard even if $k = 1$, but also, on the other hand, that the hardness proof is indeed based on the fact that the input alphabet is large (equal to the number of states in the input automaton). Showing that this problem is in NP, as well as our algorithms, requires a series of combinatorial insights on the structure of k-universal words accepted by finite automata.

Further, in Section 4, building on the aforementioned understanding of the combinatorial properties of k-universal words accepted by finite automata, we provide algorithms for counting the number of k-subsequence universal words (respectively, paths) accepted by a given deterministic (respectively, non-deterministic) finite automaton, and ranking an input word within the set of k-subsequence universal words (paths) accepted by a given deterministic (respectively, non-deterministic) finite automaton. Again, this approach extends non-trivially the approach from [2], where problems related to counting and ranking subsequence universal words (unrestricted by any regular membership constraint) were approached for the first time.

2 Preliminaries

Let $\mathbb{N} = \{1, 2, \ldots\}$ denote the natural numbers and set $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ as well as $[n] = \{1, \ldots, n\}$ and $[i, n] = \{i, i+1, \ldots, n\}$ for all $i, n \in \mathbb{N}_0$ with $i \leq n$.

An *alphabet* $\Sigma = \{1, 2, \ldots, \sigma\}$ is a finite set of symbols, called *letters* (w.l.o.g., we can assume that the letters are integers). A *word* w is a finite sequence of letters from a given alphabet and its length $|w|$ is the number of w's letters. For $i \in [|w|]$ let $w[i]$ denote w's i^{th} letter. The set of all finite words (aka strings) over the alphabet Σ, denoted by Σ^*, is the free monoid generated by Σ with concatenation as operation and the neutral element is the empty word ε, i.e., the word of length 0. Let Σ^n denote all words in Σ^* exactly of length $n \in \mathbb{N}_0$ and $\Sigma^{\leq n}$ the set of all words of Σ^* up to length $n \in \mathbb{N}_0$. Set $\text{alph}(w) = \{\mathtt{a} \in \Sigma \mid \exists i \in [|w|] : w[i] = \mathtt{a}\}$ as w's alphabet. For $u, w \in \Sigma^*$, u is called a *factor* of w, if $w = xuy$ for some words $x, y \in \Sigma^*$. If $x = \varepsilon$ (resp., $y = \varepsilon$) then u is called a *prefix* (resp., *suffix*) of w. For $1 \leq i \leq j \leq |w|$ define the factor from w's i^{th} letter to the j^{th} letter by $w[i, j] = w[i] \cdots w[j]$. Futher, given a pair of indices $i < j$, $w[j, i] = \varepsilon$. Let $<$ be an order relation on Σ (e.g., the natural order on integers). We extend this order relation to the lexicrographical order on Σ^* in the following way: the word u is lexicographically smaller than the word w ($u < w$) iff either u is a prefix of w or there exists $x, y_1, y_2 \in \Sigma^*$ and $\mathtt{a}, \mathtt{b} \in \Sigma$ with $u = x\mathtt{a}y_1$, $w = x\mathtt{b}y_2$ and $\mathtt{a} < \mathtt{b}$.

As we are interested in investigating the k-subsequence universality of regular languages, we firstly introduce the basic concepts related to subsequences. We then present the definitions for the transformation of these notions to the domain of regular languages and finite automata.

▶ **Definition 1.** *Let $w \in \Sigma^*$ and $n \in \mathbb{N}_0$. A word $u \in \Sigma^*$ is called* subsequence *of w ($u \in \text{SubSeq}(w)$) if there exist $v_1, \ldots, v_{n+1} \in \Sigma^*$ such that $w = v_1 u[1] v_2 u[2] \cdots v_n u[n] v_{n+1}$.*

Set $\mathrm{SubSeq}_k(w) = \{u \in \mathrm{SubSeq}(w) \mid |u| = k\}$.

▶ **Example 2.** Subsequences of `automatauniversality` are `auto`, `tomata`, `salty`, `mate`, and `atom` while `star` and `alien` are not because their letters do not occur in the correct order.

In [6], the authors investigated words which have, for a given $k \in \mathbb{N}_0$, all words from Σ^k as subsequence, namely k-subsequence universal words. Note that this notion is similar to the one of richness introduced and investigated in [27, 28]. We stick here to the notion of k-subsequence universality since our focus are regular languages and thus the well-known notion of the universality of automata and formal languages, i.e., $L(\mathcal{A}) = \Sigma^*$ for a given finite automaton $\mathcal{A}$, is close to the one of subsequence universality of words.

▶ **Definition 3.** *A word $w \in \Sigma^*$ is called* k-subsequence universal *(w.r.t. Σ), for $k \in \mathbb{N}_0$, if $\mathrm{SubSeq}_k(w) = \Sigma^k$. If the context is clear we briefly call w k-universal. The universality-index $\iota(w)$ is the largest k such that w is k-universal.*

We denote the set of k-universal words in a given set $\mathcal{M} \subseteq \Sigma^*$ by $\mathrm{Univ}_{\mathcal{M},k}$. Thus, the set of all k-universal words over a given alphabet Σ is denoted $\mathrm{Univ}_{\Sigma^*,k}$.

▶ **Example 4.** Consider the word $w = \mathtt{baaababb} \in \{\mathtt{a},\mathtt{b}\}^*$. We have $|\mathrm{SubSeq}_3(\mathtt{baaababb})| = |\{\mathtt{aaa},\mathtt{aab},\mathtt{aba},\mathtt{abb},\mathtt{baa},\mathtt{bab},\mathtt{bba},\mathtt{bbb}\}| = 8 = 2^3$. Thus, $\mathtt{baaababb} \in \mathrm{Univ}_{\{\mathtt{a},\mathtt{b}\}^*,3}$. Since $\mathtt{abba} \notin \mathrm{SubSeq}_4(\mathtt{baaababb})$, it follows that $\mathtt{baaababb} \notin \mathrm{Univ}_{\{\mathtt{a},\mathtt{b}\}^*,4}$ and $\iota(\mathtt{baaababb}) = 3$.

Further, we recall the *arch factorisation* by Hébrard [24] which factorises words uniquely.

▶ **Definition 5.** *The* arch factorisation *of $w \in \Sigma^*$ is given by $w = \mathrm{ar}_1(w)\cdots\mathrm{ar}_k(w)\,\mathrm{r}(w)$ for $k \in \mathbb{N}_0$ with $\iota(\mathrm{ar}_i(w)) = 1$ and $\mathrm{ar}_i(w)[|\mathrm{ar}_i(w)|] \notin \mathrm{alph}(\mathrm{ar}_i(w)[1,|\mathrm{ar}_i(w)|-1])$ for all $i \in [k]$, as well as $\mathrm{alph}(\mathrm{r}(w)) \subsetneq \Sigma$. The words $\mathrm{ar}_i(w)$ are the* arches *and $\mathrm{r}(w)$ is the* rest *of w.*

▶ **Example 6.** Continuing Example 4, we have the arch factorisation $w = (\mathtt{ba})\cdot(\mathtt{aab})\cdot(\mathtt{ab})\cdot\mathtt{b}$ where the parantheses denote the three arches and the rest `b`.

A proper subset of the k-universal words is given by all the words with an empty rest introduced in [12] as the set of *perfect k-universal* words.

▶ **Definition 7.** *We call a word $w \in \Sigma^*$* perfect k-universal *if $\iota(w) = k$ and $\mathrm{r}(w) = \varepsilon$. The set of all these words with $\mathrm{alph}(w) = \Sigma$ is denoted by $\mathrm{PUniv}_{\Sigma^*,k}$.*

▶ **Example 8.** The word `baaababb` from Examples 4 and 6 is not perfect 3-universal since it has $\iota(\mathtt{baaababb}) = 3$ but $\mathrm{r}(\mathtt{baaababb}) = \mathtt{b} \neq \varepsilon$. To give a positive example, consider $\mathtt{abcbbaccbbaacb} \in \mathrm{PUniv}_{\{\mathtt{a},\mathtt{b},\mathtt{c}\}^*,4}$ since its arch factorisation is $(\mathtt{abc})\cdot(\mathtt{bbac})\cdot(\mathtt{cbba})\cdot(\mathtt{acb})$.

▶ **Theorem 9** ([6]). *Let $w \in \Sigma^{\geq k}$ with $\mathrm{alph}(w) = \Sigma$. Then we have $\iota(w) = k$ iff w has exactly k arches.*

In the remainder of this section, we introduce the basic definitions we need to define the k-universality of regular languages. For basic notions on finite automata, we refer to [26]. A *non-deterministic finite automaton (NFA)* $\mathcal{A}$ is a tuple $(Q, \Sigma, q_0, \delta, F)$ with the finite set of states Q (of cardinality $n \in \mathbb{N}$), an initial state $q_0 \in Q$, the set of final states $F \subseteq Q$, an input alphabet Σ, and a transition function $\delta : Q \times \Sigma \to 2^Q$, where 2^Q is the powerset of Q. If we have $|\delta(q,\mathtt{a})| = 1$ for all $q \in Q, \mathtt{a} \in \Sigma$, then $\mathcal{A}$ is called *deterministic* (DFA). We call a sequence $\pi = (q_0, \mathtt{a}_1, q_1, \mathtt{a}_2, \ldots, \mathtt{a}_\ell, q_\ell)$ an ℓ-length path in $\mathcal{A}$ iff $q_i \in Q$ for all $i \in [0,\ell]$ and $q_{i+1} \in \delta(q_i, \mathtt{a}_{i+1})$, for all $i \in [0, \ell-1]$. The word $w_\pi = \mathtt{a}_1\cdots\mathtt{a}_\ell$ is the *word (label) associated to* π. A path is simple if it does not contain the same state

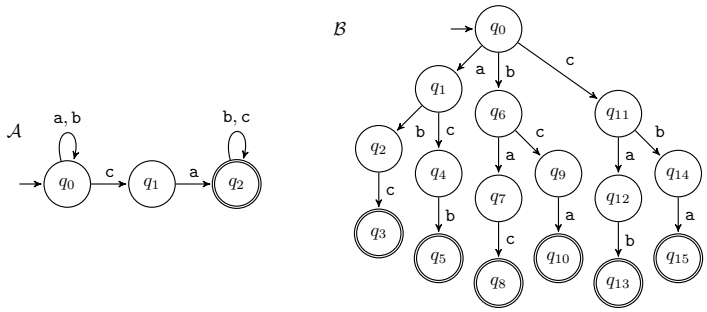


Figure 1 A 2-∃-universal NFA $\mathcal{A}$ and a 1-∀-universal NFA $\mathcal{B}$.

twice. Moreover, a state $q \in Q$ is called accessible (respectively, co-accessible) in $\mathcal{A}$ if there exists a path connecting q_0 to q (respectively, q to a final state). A path is called *accepting* if $q_\ell \in F$ holds. Define the language of $\mathcal{A}$, i.e., the set of words *accepted* by $\mathcal{A}$, by $L(\mathcal{A}) = \{w \in \Sigma^* \mid \exists \text{ accepting path } \pi \text{ in } \mathcal{A} : w = w_\pi\}$. For abbreviation, we set $\mathcal{A}_n = L(\mathcal{A}) \cap \Sigma^n$ and $\mathcal{A}_{\leq n} = L(\mathcal{A}) \cap \Sigma^{\leq n}$. Note that the class of languages accepted by NFAs is equal to the class of languages accepted by DFAs and it is equal to the class of regular languages. Moreover, for every word $w \in L(\mathcal{A})$ there exists exactly one (in the deterministic case) or a set of (in the non-deterministic case) associated path(s). For a word w accepted by a finite automaton, let π_w denote one accepting path labelled with w; in the case when there are multiple such paths, we simply choose one of them. Since we usually are interested in only one path for a word $w \in L(\mathcal{A})$, we refer to it as π_w.

▶ **Definition 10.** *For a given NFA $\mathcal{A}$, the* reverse transition function $\Delta(q, \mathtt{x})$ *returns the set of states in Q with a transition labelled by* $\mathtt{x}$ *to q, i.e.,* $\Delta(q, \mathtt{x}) = \{q' \in Q \mid \delta(q', \mathtt{x}) = q\}$.

Now, we can define the k-subsequence universality of a regular language L. Here, we distinguish whether at least one or all words of L are k-universal w.r.t. Definition 3. Note that we always take the minimal alphabet Σ such that $L \subseteq \Sigma^*$ as a reference when considering the k-universality of words from L.

▶ **Definition 11.** *Let L be a regular language. L is called* existence k-subsequence universal *(k-∃-universal) for some $k \in \mathbb{N}$, if there exists a k-universal word $w \in L$. L is called* universal k-subsequence universal *(k-∀-universal) for some $k \in \mathbb{N}$, if all $w \in L$ are k-universal.*

If a regular language L, accepted by some finite automaton $\mathcal{A}$ (NFA or DFA), is k-∃-universal (respectively, k-∀-universal) then we also say that the automaton $\mathcal{A}$ is k-∃-universal (respectively, k-∀-universal). Further, we also say that a *path π in $\mathcal{A}$ is k-universal* if the label of π, namely w_π, is k-universal. As an example, a 2-∃-universal NFA $\mathcal{A}$, which is not 3-∃-universal, and a 1-∀-universal NFA $\mathcal{B}$ which is not 2-∀-universal are shown in Figure 1 (note that $\mathcal{B}$ recognises exactly every permutation of $\mathtt{abc}$). Based on this definition we define two associated decision problems.

▶ **Problem 12.** The *existence subsequence universality problem for regular languages (k-ESU)* is to decide for a regular language, given by a finite automaton $\mathcal{A}$ recognising it, and $k \in \mathbb{N}$, given in binary representation, whether L is k-∃-universal.

▶ **Problem 13.** The *universal subsequence universality problem for finite automata (k-ASU)* is to decide for a regular language, given by a finite automaton $\mathcal{A}$ recognising it, and $k \in \mathbb{N}$, given in binary representation, whether L is k-∀-universal.

We finish this section by introducing the *rank* of a word in order to enumerate the k-universal words accepted by a given finite automaton.

▶ **Definition 14.** *The* rank *of a word w within the set* $\mathrm{Univ}_{\mathcal{M},k}$ *is the number of words in* $\mathrm{Univ}_{\mathcal{M},k}$ *lexicographically smaller than w, i.e. $rank(w) = |\{v \in \mathrm{Univ}_{\mathcal{M},k} \mid v < w\}|$.*

▶ Remark 15. By Definition 3, the set of all k-universal words accepted by $\mathcal{A}$ is given by $\mathrm{Univ}_{L(\mathcal{A}),k}$, the set of all n-length k-universal words accepted by $\mathcal{A}$ is given by $\mathrm{Univ}_{\mathcal{A}_n,k}$, and the set of all k-universal words of length at most n accepted by $\mathcal{A}$ is given by $\mathrm{Univ}_{\mathcal{A}_{\leq n},k}$.

The computational model we use is the standard unit-cost RAM with logarithmic word size: for an input of size n, each memory word can hold $\log(n)$ bits. Arithmetic and bitwise operations with numbers in $[1, n]$ are, thus, assumed to take $O(1)$ time. Numbers larger than n, with ℓ bits, are represented in $O(\ell/\log n)$ memory words, and working with them takes time proportional to the number of memory words on which they are represented. In all the problems, we assume that we are given a number k, binary encoded, and one finite automaton $\mathcal{A}$, specified as the set of states, set of input letters, set of transitions, and initial and final states. The size of the input is, as such, the size of the binary encoding of k, which is $\lceil \log_2 k \rceil$, plus the size S of the encoding of $\mathcal{A}$ (which is lower bounded by the number of edges in the graph associated to the automaton). So, one memory word can hold $\log(S + \log_2(k))$ bits.

For a more detailed general discussion on this model see, e. g., [8].

Some of our algorithms run in exponential time and use exponential space w.r.t. the size n of the input. Following the literature dealing with such exponential algorithms (see, e.g., [15]), we will use the O^*-notation. By definition, for functions f and g we write $f(n) = O^*(g(n))$ if $f(n) = O(g(n)n^{O(1)})$. In other words, the O^*-notation hides polynomial factors, just as the O-notation hides constants. Using this notation, we can assume that our single-exponential time and single-exponential space algorithms run on a RAM model where arithmetic and bitwise operations with single exponential numbers (w.r.t. the size n of the input) as well as accessing the memory-words (given their address, like in an usual RAM) are assumed to take $O^*(1)$ time. Working with such a computational model allows us to analyse the actual algorithms rather than the various intricacies of the computational model.

In particular, when expressing the complexity of our algorithms, we get functions which are exponential in σ (the size of the alphabet) but polynomial in the number n of states of the input NFA or, for the enumeration algorithms, in the length m of the enumerated strings or in the value of the input number k (the parameter of the considered problems). For clarity of the exposure, although we use the O^*-notation, we will explicitly write the dependency on n, m, and k, and only hide the polynomial dependency on σ.

3 Decision Problems

In this section, we consider the decision problems k-ESU and k-ASU. We begin with a series of combinatorial observations.

▶ Remark 16. Every path acccepted by an NFA $\mathcal{A}$ is k-universal iff every simple accepting path in it is k-universal. In one direction, if $\mathcal{A}$ accepts any path that is not k-universal, then not every path accepted by $\mathcal{A}$ is k-universal. Otherwise, the set of paths induced by every word accepted by $\mathcal{A}$ must contain, as a subsequence, some non-cyclic path. Therefore, if each non-cyclic path is k-universal, then every path is.

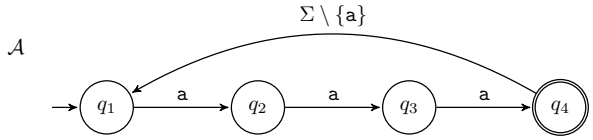


Figure 2 Note that the shortest k-universal word accepted by $\mathcal{A}$ has length $(\sigma-1)kn$.

▶ **Lemma 17.** *Let $q, q' \in Q$ be a pair of states in the NFA $\mathcal{A}$ such that there exists a path π from q to q' where the final transition in π is labelled $\mathtt{x} \in \Sigma$. Then, there exists some path π' of length at most $n = |Q|$ from q to q' where the final transition in the path is labelled $\mathtt{x}$.*

Proof. Let $\hat{q} \in Q$ be the state in π before q', i.e., the state such that $q' \in \delta(\hat{q}, \mathtt{x})$. As there are at most n states in $\mathcal{A}$, given any pair of states $q_1, q_2 \in Q$, if q_2 can be reached from q_1, then there must exist a path from q_1 to q_2 of length at most $n-1$. In particular, there exists a path of length $n-1$ from q to $\hat{q}$. Extending this path by the transition from $\hat{q}$ reading $\mathtt{x}$ to q', we obtain a path π' from q to q' of length at most n. ◀

▶ **Lemma 18.** *For an NFA $\mathcal{A}$ with n states, if there is a k-universal word accepted by $\mathcal{A}$, then there is a k-universal word accepted by $\mathcal{A}$ of length at most $kn\sigma - (n-1)(k-1)$.*

Proof. Let w be a k-universal word accepted by $\mathcal{A}$ and for all $i \in [k]$ let $a_{i,1}, \dots a_{i,\sigma} \in \Sigma$ be the letters of Σ in the order they occur in $\mathrm{ar}_i(w)$, that is $a_{i,1} = \mathrm{ar}_i(w)[1]$ and $a_{i,j} = \mathrm{ar}_i(w)[\ell]$ such that $j - 1 = |\operatorname{alph}(\mathrm{ar}_i(w)[1, \ell-1])| < |\operatorname{alph}(\mathrm{ar}_i(w)[1,\ell])| = j$ for $1 < j \leq \sigma$. Then $\mathrm{ar}_i(w) = a_{i,1}u_{i,1}a_{i,2}\cdots u_{i,\sigma-1}a_{i,\sigma}$ where $u_{i,j} \in \Sigma^*$ is a word such that there is a path $\pi_{u_{i,j}}$ in $\mathcal{A}$ labeled with $u_{i,j}$. For the sake of readability we denote the suffix of w starting after $\mathrm{ar}_k(w)$ by $u_{k+1,1}$ in this proof. By Lemma 17, $u_{i,j}$ is accepted by an automaton with at most n states obtainable from $\mathcal{A}$ by changing the initial (respectively, final) state of $\mathcal{A}$ to the starting (respectively, end) state of $\pi_{u_{i,j}}$. Hence we can find a word $v_{i,j}$ of length at most $n-1$ which is the label of a path starting and ending in the same state as $\pi_{u_{i,j}}$. Let w' be the word obtained from w by replacing every $u_{i,j}$ by $v_{i,j}$. Then w' is a k-universal word accepted by $\mathcal{A}$ of length $|w'| = \sum_{i=1}^{k}\left(\sum_{j=1}^{\sigma}|a_{i,j}| + \sum_{j=1}^{\sigma-1}|v_{i,j}|\right) + |v_{k+1,1}| \leq k\left(\sigma + (\sigma-1)(n-1)\right) + n - 1 = kn\sigma - (k-1)(n-1)$ (cf. Figure 2). ◀

We now move to the main results of this section and give fixed parameter tractable (FPT) algorithms w.r.t. σ. The time complexity of these algorithms is polynomial for $\sigma \in O(\log n)$ and does not depend at all on k.

▶ **Lemma 19.** *For a given NFA $\mathcal{A}$ with n states and $|\Sigma| = \sigma$, we can decide in $O^*(n^3 2^\sigma)$ time whether $\mathcal{A}$ accepts words whose universality index is arbitrarily large. If the answer is negative, then we can compute the largest universality index of a word accepted by $\mathcal{A}$.*

Proof. Assume that $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, as defined in Section 2. Let us assume w.l.o.g. that $\mathcal{A}$ contains only accessible and co-accessible states.

Our approach is based on a series of observations. We first make these observations, and then provide an algorithm proving the statement of the lemma.

Note first that if there exists a state $q \in Q$ (which is both accessible and co-accessible, by assumption) such that there is a path in $\mathcal{A}$ from q to q labelled with a word which contains all letters of Σ, then we can immediately decide that $\mathcal{A}$ accepts words whose universality index is arbitrarily large. Indeed, to obtain an accepted word (accepting path), whose universality index is at least ℓ, we follow the path from q_0 to q, then follow ℓ times the path which

contains all letters of Σ going from q to q, and finally follow a path connecting q to a final state.

Secondly, assume that there exists no state $q \in Q$ such that there is a path in $\mathcal{A}$ from q to q labelled with a word which contains all letters of Σ. In this setting we note that, for each state $q \in Q$, there exists a unique maximal (w.r.t. inclusion) subset $V_q \subsetneq \Sigma$ such that there exists a path from q to q labelled with a word β_q with $\mathrm{alph}(\beta_q) = V_q$. Indeed, if two such different maximal sets V_q' and V_q'' would exist, witnessed by the words w' and w'', then we can follow the path labelled with $w'w''$, which goes from q to q, and $\mathrm{alph}(w'w'') = V_q' \cup V_q''$ (and this includes both V_q' and V_q'').

Finally, consider an accepting path $\pi = (q_0, \mathtt{a}_1, q_1, \mathtt{a}_2, \ldots, \mathtt{a}_\ell, q_\ell)$ of $\mathcal{A}$. We can rewrite the path π as follows:

- Find the rightmost occurrence of q_0 in π; let us assume that this is the h^{th} state on this path, namely q_h. Replace the subpath connecting the initial occurrence of q_0 to q_h with the loop $(q_0, \mathtt{a}_1 \cdots \mathtt{a}_h)^\circ$ (where we use $\circ$ to emphasise this is a loop). Now the path is rewritten as $((q_0, \mathtt{a}_1 \cdots \mathtt{a}_h)^\circ, \mathtt{a}_{h+1}, q_{h+1}, \ldots, \mathtt{a}_\ell, q_\ell)$. If q_0 appears only once in π, then the path is rewritten as $((q_0, \varepsilon)^\circ, \mathtt{a}_1, q_1, \ldots, \mathtt{a}_\ell, q_\ell)$.
- Now, we repeat the procedure for the path $(q_{h+1}, \mathtt{a}_{h+2}, \ldots, \mathtt{a}_\ell, q_\ell)$ (respectively, if q_0 appeared only once on π, for the path $(q_1, \ldots, \mathtt{a}_\ell, q_\ell)$).

After completing this process, one obtains a *normal-form representation* of the path π as $((q_0', \alpha_1)^\circ, \mathtt{a}_1', (q_1', \alpha_2)^\circ, \mathtt{a}_2', \ldots, \mathtt{a}_r', (q_r', \alpha_{r+1})^\circ)$, with $r \leq n$, as the states $q_0' = q_0, q_1', \ldots, q_r'$ are pairwise distinct; moreover, $\alpha_i \in \Sigma^*$ for all $i \in [r]$.

Let us come back now to the case when there exists no state $q \in Q$ such that there is a path in $\mathcal{A}$ from q to q labelled with a word which contains all letters of Σ. Consider now all accepting paths $((q_0', \alpha_1)^\circ, \mathtt{a}_1', (q_1', \alpha_2)^\circ, \mathtt{a}_2', \ldots, \mathtt{a}_r', (q_r', \alpha_{r+1})^\circ)$ (given in normal-form representation) where the states $q_0', \ldots, q_r'$ and the letters $\mathtt{a}_1', \ldots, \mathtt{a}_r'$ are fixed, and the loops $\alpha_1, \ldots, \alpha_r$ are variable. We are interested in how we can, for $i \in [r+1]$, choose $\alpha_i \in V_{q_{i-1}}^*$ in order to maximise the universality index of the resulting word. We claim that it is enough to take $\alpha_i = \beta_{q_{i-1}}^2$ (where the words β_q were defined above). Indeed, this holds because the arch decomposition of the word labelling the path $((q_0', \alpha_1)^\circ, \mathtt{a}_1', (q_1', \alpha_2)^\circ, \mathtt{a}_2', \ldots, \mathtt{a}_r', (q_r', \alpha_{r+1})^\circ)$ is done greedily from left to right, and we are thus interested in packing as many arches as possible in each of the prefixes $((q_0', \alpha_1)^\circ, \mathtt{a}_1', (q_1', \alpha_2)^\circ, \mathtt{a}_2', \ldots, \mathtt{a}_i', (q_i', \alpha_{i+1})^\circ)$, for all i. So, we begin by noting that the alphabet of the word labelling $((q_0', \alpha_1)^\circ)$ is always included in $V_{q_0'}$, and is indeed equal to $V_{q_0'}$ for, e.g., $\alpha_1 = \beta_{q_0'}^2$. Now, let us consider $i \geq 1$ and the path $((q_0', \alpha_1)^\circ, \mathtt{a}_1', (q_1', \alpha_2)^\circ, \mathtt{a}_2', \ldots, \mathtt{a}_i', (q_i', \alpha_{i+1})^\circ)$. We note that, as we do not have 1-universal loops, the loop $(q_i', \alpha_{i+1})^\circ$ can at most complete the final (and previously incomplete) arch of $((q_0', \alpha_1)^\circ, \mathtt{a}_1', (q_1', \alpha_2)^\circ, \mathtt{a}_2', \ldots, \mathtt{a}_{i-1}', (q_{i-1}', \alpha_i)^\circ)$; after this potential completion, we can only try to add as many new letters as we can in the last arch. So, we will define α_{i+1} as a power of the word $\beta_{q_i'}$, that labels the loop containing q_i' which adds the most new letters to the constructed word, and moreover it is enough to only use $\beta_{q_i'}$ twice in this power (i.e., $\alpha_{i+1} = \beta_{q_i'}^2$). To see that this holds, note that if the rest of the word labelling the path $((q_0', \alpha_1)^\circ, \mathtt{a}_1', (q_1', \alpha_2)^\circ, \mathtt{a}_2', \ldots, \mathtt{a}_{i-1}', (q_{i-1}', \alpha_i)^\circ)$ is completed to a new arch by a repetition $\beta_{q_i'}^f$ for some $f > 2$, then this is done by the first $\beta_{q_i'}$ from this repetition. Then, the suffix $\beta_{q_i'}^{f-1}$ adds as many new letters to the alphabet of the rest of the resulting word as a single occurrence of $\beta_{q_i'}$. This completes the proof of our claim.

In conclusion: among all accepting paths $((q_0', \alpha_1)^\circ, \mathtt{a}_1', (q_1', \alpha_2)^\circ, \mathtt{a}_2', \ldots, \mathtt{a}_r', (q_r', \alpha_{r+1})^\circ)$ (given in normal-form representation) the path $((q_0', \beta_{q_0'}^2)^\circ, \mathtt{a}_1', (q_1', \beta_{q_1'}^2)^\circ, \mathtt{a}_2', \ldots, \mathtt{a}_r', (q_r', \beta_{q_r'}^2)^\circ)$ has the highest universality index.

Based on these observations, we can now state the main idea of our algorithm. We first

preprocess the input automaton to get rid of the states which are not accessible and of the states which are not co-accessible. Then, we check whether there exists a state $q \in Q$ such that there is a path in $\mathcal{A}$ from q to q labelled with a word which contains all letters of Σ; if yes, we simply decide that there exists k-universal accepted words, for any k. If no such state exists, we compute, by dynamic programming, the path $((q'_0, \beta^2_{q'_0})^\circ, \mathtt{a}'_1, (q'_1, \beta^2_{q'_1})^\circ, \mathtt{a}'_2, \ldots, \mathtt{a}'_r, (q'_r, \beta^2_{q'_r})^\circ)$ with the highest universality index, for all states $q_r \in Q$.

In the following we describe this algorithm in more details, by highlighting its main phases.

Preprocessing: We determine the accessible states of $\mathcal{A}$ by running a graph traversal algorithm on $\mathcal{A}$ starting from the state q_0. The not-accessible states are removed. We determine the co-accessible states of $\mathcal{A}$ by running a graph traversal algorithm on the graph of $\mathcal{A}$, with the direction of all edges inverted, starting from the final states. The not-co-accessible states of $\mathcal{A}$ are removed. The complexity of this step is $O(n^3)$ in the worst case. Note that for this step it is enough to consider a simplified form of the graph associated to $\mathcal{A}$, where we only see, for each two states, if there is at least one edge between them or not; as such, the total size of the graph $\mathcal{A}$ is $O(n^2)$). From now on, we assume that all states of $\mathcal{A}$ are both accessible and co-accessible.

Universal Loops: In this step, we determine whether there exists a state q of $\mathcal{A}$ such that there is a path in $\mathcal{A}$ from q to q labelled with a word which contains all letters of Σ. This is done as follows. For each state $q \in Q$, we run the following process. We define L to be a queue, initially containing the element $(q, \emptyset)$, and S be a set which is initially empty. As long as L is not empty, we pop the first element from L, let it be (q', R'), and update S to $S \cup \{(q', R')\}$. Now, for all transitions leaving q of the form $q'' \in \delta(q', \mathtt{a})$, if $(q'', R' \cup \{\mathtt{a}\}) \notin S$, we insert $(q'', R' \cup \{\mathtt{a}\})$ into L. When L is empty, we are done, and we simply check if (q, Σ) is contained in S. If (q, Σ) is contained in S then there exists a path from q to q which is 1-universal. Otherwise, there is no such path. Moreover, we also compute and store the set V_q of maximal cardinality for which (q, V_q) belongs to S.

After running the above process for all states in Q, if we have identified a state q for which there exists a path from q to q which is 1-universal, then we simply conclude that $\mathcal{A}$ accepts words whose universality index is arbitrarily large. Otherwise, we continue with the next phase.

The complexity of this phase is $O^*(n^3 2^\sigma)$, as the numbers of transitions leaving a state q is upper bounded by $n\sigma$.

No Universal Loop: In this case, we will use a $n \times 2^\sigma$ matrix $M[\cdot][\cdot]$ (and an auxiliary matrix M' of the same size). Our algorithm has at most n iterations, and we will maintain the property that, before iteration $\ell+1$ (for $\ell \geq 0$), $M[q'_r][V]$ is the maximum among the number of arches of some path with the normal-form $((q'_0, \beta^2_{q'_0})^\circ, \mathtt{a}'_1, (q'_1, \beta^2_{q'_1})^\circ, \mathtt{a}'_2, \ldots, \mathtt{a}'_r, (q'_r, \beta^2_{q'_r})^\circ)$, with $q'_0 = q_0$ and $r \leq \ell$, such that the rest $\mathrm{r}(u)$ of the word u labelling this path has the alphabet V, or $M[q'_r][V] = -1$ if no such path exists.

To begin with, before the first iteration of the loop, we initialize $M[q][V] = -1$ for all q and V. Then we set $M[q_0][V_{q_0}] = 0$.

In the ℓ^{th} iteration, we copy M into the matrix M'. We then go through all q and V and, if $M[q][V] \neq -1$, we do the following: for all letters $a \in \Sigma$, for all $q' \in \delta(q, a)$, set $V' = V \cup \{a\}$ and $\delta = 0$. If $V' = \Sigma$, we reset $V' = \emptyset$ and set $\delta = 1$. Then, we compute $V'' = V' \cup V_{q'}$. If $V'' = \Sigma$ (note that this can only happen when $V' \neq \emptyset$) we reset $V'' = \emptyset$ and set $\delta = 1$. Finally, we compute $V''' = V'' \cup V_{q'}$ and set $M'[q][V'''] = \max\{M'[q'][V''], M[q][V] + \delta\}$.

We stop this process after n iterations.

Clearly, before the first iteration of our algorithm, the property we intend to maintain holds. Then, in the i^{th} iteration, we try to extend the already constructed path ending in state q, with rest V, by reading first one letter and reaching state q', and then by reading $\beta^2_{q'}$, and compute the universality index of this resulting path. So, the property is maintained in the ℓ^{th} iteration.

The complexity of this phase is $O^*(n^3 2^\sigma)$, if all the above steps are implemented naïvely.

Now, the highest universality index of a word accepted by $\mathcal{A}$ is the largest value $M[q][V]$ for $q \in F$, $V \subseteq \Sigma$.

Conclusion: The algorithm consisting of the three phases above decides whether $\mathcal{A}$ accepts words whose universality index is arbitrarily large, or, if this is not the case, computes the largest universality index of a word accepted by $\mathcal{A}$. Its time complexity is $O^*(n^3 2^\sigma)$. ◀

As a corollary of Lemma 19, we get the following theorem.

▶ **Theorem 20.** *For a given NFA $\mathcal{A}$ with n states and $|\Sigma| = \sigma$ and a natural number $k \in \mathbb{N}$, we can decide k-ESU in $O^*(n^3 2^\sigma)$ time.*

Proof. We first use Lemma 19 to test whether $\mathcal{A}$ accepts words whose universality index is arbitrarily large. If yes, then the answer for the given instance of k-ESU is positive. If $\mathcal{A}$ does not accept words whose universality index is arbitrarily large, we compute the largest universality index ℓ of a word accepted by $\mathcal{A}$. If $\ell \geq k$, then the answer for the given instance of k-ESU is positive. Otherwise, the answer for the given instance of k-ESU is negative. ◀

Next, we approach the k-ASU problem. Once again, we show a preliminary lemma.

▶ **Lemma 21.** *For a given NFA $\mathcal{A}$ with n states and $|\Sigma| = \sigma$, we can compute in $O^*(n^3 2^\sigma)$ time the smallest universality index of a word accepted by $\mathcal{A}$.*

Proof. Clearly, the set of words accepted by $\mathcal{A}$ which have the smallest universality index (among all words accepted by $\mathcal{A}$) includes a word which is the label of a simple path in $\mathcal{A}$. So, to solve the problem stated in this lemma it is enough to consider only words which are the label of simple paths in $\mathcal{A}$. The length of these words is upper bounded by n. This also shows that their universality index is upper bound by n (so, as a consequence, this universality index and the numbers smaller than it fit in one memory word).

The solution of our problem is done by a dynamic programming algorithm. We define two $n \times 2^\sigma$ matrices M, M' where initially $M[q][V] = M'[q][V] = \infty$ for $V \subseteq \Sigma$. Also, we use a set $L = \emptyset$. Further, set $M[q_0][\emptyset] = 0$ and insert $(q_0, \emptyset)$ in L.

We will maintain the property that before the $(r+1)^{\text{th}}$ iteration of our algorithm (for $r \geq 0$) $M[q][S]$ stores the minimal number of arches of the word $w_{r,q}$ of length at most r which connects q_0 and q such that $\text{alph}(\text{r}(w_{r,q})) = S$. This property clearly holds before the first iteration of the algorithm. Now, we perform the following steps

```
for ℓ = 1 to n
  for all (q, S) ∈ L
    for all a ∈ Σ and q' ∈ δ(q, a)
      if S ∪ {a} = Σ
        set M'[q'][∅] = M[q][S] + 1 and insert (q', ∅) in L
      else
        set M'[q'][S ∪ {a}] = M[q][S] and insert (q', S ∪ {a}) in L
for all q ∈ Q and V ⊆ Σ
  set M[q][V] = min{M[q][V], M'[q][V]}
```

In the iteration of the outmost for-loop for $\ell = r$ we actually consider, for each pair (q, S), the word v of length at most $r-1$ which connects q_0 and q, such that v has a minimum number of arches and $\text{alph}(\text{r}(v)) = S$ (the relevant information about this path, i.e., its rest and the number of arches is stored in M), and try to extend this path by one letter $\mathtt{a}$. This leads to another pair $(q', S \cup \{\mathtt{a}\})$, and we simply check if this is the word v' with a minimum number of arches, of length at most r which connects q_0 and q', such that $\text{alph}(\text{r}(v'))$ is either $\emptyset$, when $\Sigma = S \cup \{\mathtt{a}\}$, or $S \cup \{\mathtt{a}\}$ otherwise (see also Lemma 25, which explains how a path is extended). This implies that the property which we wanted to maintain holds before the iteration of the loop for $\ell = r + 1$.

The time complexity of this algorithm is $O^*(n^3 2^\sigma)$.

Now, the smallest universality index of a word accepted by $\mathcal{A}$ is simply the minimum entry $M[q][V]$, for a final state q. ◀

▶ **Theorem 22.** *For a given NFA $\mathcal{A}$ with n states, and input alphabet of size σ, and a natural number k, k-ASU is decidable in $O^*(n^3 2^\sigma)$ time.*

Proof. We first use Lemma 21 to compute ℓ, the smallest universality index of a word accepted by A. If $\ell < k$, then the answer for the given instance of k-ASU is negative. Otherwise, the answer is positive. ◀

After the submission of this conference paper, we realized that the algorithmic problems solved in Lemma 21 and Theorem 22 can actually be solved, in fact, in polynomial time. These results will be included in an extended version of this paper.

We conclude this section by showing that k-ESU is actually NP-complete, and it is NP-hard even for $k = 1$. Clearly, in the light of our previous results from Theorem 20, for this problem to be NP-hard we need to consider the case of an input alphabet $\sigma \in \Omega(\log n)$.

▶ Remark 23. Note that an automaton accepts a word w which is k-universal for $k > n = |Q|$ iff there exists a state q, which is both accessible and co-accessible, and a loop from q to q labelled with a ℓ-universal word, for some $\ell \geq 1$. Indeed, for the left-to-right implication, we look at the states q_i reached by reading the first i arches of the word w, for $i \in [k]$. Clearly, there will be some $i < j$ such that $q_i = q_j$, and the subpath between q_i and q_j on the path labelled with w is ℓ-universal for some $\ell \geq 1$. The other implication is immediate.

▶ **Theorem 24.** *k-ESU is NP-complete.*

Proof. We begin by showing that this problem is in NP.

To solve k-ESU in NP-time, we first check if the automaton $\mathcal{A}$ contains a state q, which is both accessible and co-accessible, and a path from q to q labelled with a x-universal word. First, we traverse (deterministically) the graph of $\mathcal{A}$ from q_0 to determine the accessible states, and also from the final states using the inverted edges to determine the co-accessible states. For each of these states, we non-deterministically guess a 1-universal word of length at most $n\sigma$ and see if this takes us from q to q (by Lemma 18 we have that, if there exists a 1-universal word accepted by an altered version of the automaton $\mathcal{A}$, where q is the unique initial and final state, then there exists one such word of length at most $n\sigma$). If we successfully guessed this word, and $k > n$, then we simply accept the input automaton; otherwise, we reject. If $k \leq n$ and no such altered autonaton exists, then follwoing Lemma 18 any k-subsequence universal word must have length at most $kn\sigma$. Therefore, we can check every word in $\Sigma^{kn\sigma}$ to determine if any word is both k-subsequence universal and accepted by $\mathcal{A}$. If such a word is found we accept the input automaton, otherwise we reject.

The fact that the problem is NP-hard follows from the proof of Theorem 3 from [29]. For completeness (as the respective proof is not given in the accessible version of the paper), we sketch here a reduction from the Hamiltonian Path Problem. The high level idea behind this

reduction is to take a graph $G = (V, E)$ containing n vertices $v_1, \ldots, v_n$ and construct an automaton $\mathcal{A}$ containing $n^2 + 2$ states, with a unique starting state q_0, a unique failure state q_f, and a set of n^2 states labeled $q_{i,j}$ for every $i, j \in [1, n]$. The state $q_{i,j}$ is used to represent visiting the vertex v_j at the i^{th} step of some path in G. With this in mind, a transition exists from $q_{i,j}$ to $q_{\ell,k}$ if and only if $\ell = i + 1$, and (j, k) is an edge in G. In order to map the paths accepted by this automaton directly to the paths in G, each transition is labelled by the index k corresponding to the end state of the edge, i.e., the transition between $q_{i,j}$ and $q_{i+1,k}$ is labelled by k. With this construction, the path in $\mathcal{A}$ labelled by the word w corresponds directly to some path of length $|w|$ in G.

As a Hamiltonian path must have length exactly n, for every $j \in [1, n]$ the state $q_{n,j}$ is marked as an accepting state, and every transition from $q_{n,j}$ leads to the failure state q_f. From this construction, any 1-universal word must correspond exactly to some permutation of the alphabet $[n]$, representing a Hamiltonian path in G. In the other direction if no such word exists, then there does not exist any such path. ◀

As far as k-ASU is concerned, from Lemma 16, and the other results presented here, we can only infer that this problem is in coNP. However, as mentioned above, more recent results show that it can actually be solved in polynomial time.

4 Counting and Ranking

In this section we discuss the problems of counting and ranking efficiently k-universal words from a regular language, given as a DFA, or k-universal paths in the case when the respective language is given as an NFA. Note, that there is a one-to-one correspondence between paths and words in the case of DFAs, so, for the sake of simplicity, from now on we will simply talk about counting and ranking paths accepted by the finite automata we are given as input.

Here, we define the problem of *counting* and *ranking* problems, for a given automaton $\mathcal{A}$, and universality index k. The counting problem is defined as the problem of determining the number of k-universal words accepted by $\mathcal{A}$, equivalent to determining the size of $\mathrm{Univ}_{L(\mathcal{A}),k}$. For a given length $m \in \mathbb{N}$, the problem of counting the number of k-universal words of length exactly (respectively, at most) m is the problem of determining the size of $\mathrm{Univ}_{\mathcal{A}_m,k}$ (respectively $\mathrm{Univ}_{\mathcal{A}_{\leq m},k}$. The *rank* of a word $w \in \mathcal{A}_n$ is the number of k-universal words accepted by $\mathcal{A}$ that are smaller than w, i.e. the size of $\{v < w \mid v \in \mathrm{Univ}_{L(\mathcal{A}),k}\}$. For a given length m, the *rank* of w within the set of k-universal words of length exactly (respectively, at most) m is the problem of determining the size of the set $|\{v < w \mid v \in \mathrm{Univ}_{\mathcal{A}_m,k}\}|$ (respectively, $|\{v < w \mid v \in \mathrm{Univ}_{\mathcal{A}_{\leq m},k}\}|$).

Recall that we are operating on a DFA or NFA $\mathcal{A}$ with n states and an alphabet of size $\sigma > 1$ (the case $\sigma = 1$ is trivial). We are also given as input the natural numbers k and m in binary representation: we are interested in counting (ranking) the k-universal words of length m contained in $L(\mathcal{A})$. It is important to know that these problems are only interesting for $k \leq m/\sigma$; otherwise, there are no m-length k-universal words. However, an additional difficulty related to this problem, compared to the case of the decision problems discussed in Section 3, is that we need to do arithmetics with large numbers. In general, if M is an upper bound on the value of the numbers that we need to process and ω is the size of the memory word of our model, then each arithmetic operation requires at most $O(\frac{\log M}{\omega})$ time to complete. In the cases we approach here, $M \leq (n\sigma)^{m+1}$ (a crude upper bound on the total number of paths of length at most m in $\mathcal{A}$), so each arithmetic operation requires at most $O(\frac{(m+1)\log(n\sigma)}{\omega})$, that is $O^*(m)$ time.

This section is laid out as follows. First, we consider the problems of counting the number of k-universal accepting paths of the finite automaton $\mathcal{A}$. For a given $m \in \mathbb{N}$ this is split into two cases, counting the number of k-universal accepting paths of $\mathcal{A}$ of length exactly m, and counting the number of k-universal accepting paths of $\mathcal{A}$ of length at most m. We show that both can be computed in $O^*(m^2n^2k2^\sigma)$ time. Additionally, we show that the number of k-universal accepting paths of $\mathcal{A}$ can be computed in $O^*(n^4k^22^\sigma)$ time.

We extend our counting results to the ranking setting, showing that a path π (respectively, word w) can be ranked within the set of k-universal paths (respectively, words) of length exactly m accepted by the NFA (respectively, DFA) $\mathcal{A}$ in $O^*(m^2n^2k2^\sigma)$ time, of length at most m in $O^*(m^2n^2k2^\sigma)$ time, and of any length in $O^*(n^4k^22^\sigma)$ time.

The main tool used in this section is the $n \times (m+1) \times k \times 2^\sigma$ size table T, which we refer to as the *path table of length* m for a given $m \in \mathbb{N}_0$. Each entry in the table T is indexed by a state $q \in Q$, a length $\ell \in [0, m]$, the number of arches $c \in [0, k-1]$, and a subset of symbols $\mathcal{R} \subset \Sigma$. The entry $T[q, \ell, c, \mathcal{R}]$ contains the number of ℓ-length paths starting at the state q_0 and ending in the state q such that the words induced by the paths contain each c arches, and the alphabet of the rest is $\mathcal{R}$. Thus, in the context of the arch factorisation, we are interested in all words belonging to paths in $\mathcal{A}$ which are *not yet* k-universal. Formally, we have $T[q, \ell, c, \mathcal{R}] = \sum_{w \in \Sigma^\ell, \mathrm{alph}(\mathrm{r}(w)) = \mathcal{R}, \iota(w) = c} |\mathcal{P}(w, q)|$, where $\mathcal{P}(w, q)$ is the set of paths from q_0 to q in $\mathcal{A}$ labelled by w. Note, that for DFAs, we have $|\mathcal{P}(w, q)| = 1$. The words which have at least k arches in their arch factorisation are captured in the auxiliary $n \times m+1$ size table $U[q, \ell]$, which we refer to as the *universal words table*. Each entry in U is indexed by a state $q \in Q$ and length $\ell \in [0, m]$ with the entry $U[q, \ell]$ containing the number of ℓ-length paths ending at q which are k-universal, i.e., $\iota(w) \geq k$.

The remainder of this section provides the combinatorial and technical tools needed to construct the tables T and U. Theorems 29 and 34, and Corollary 31 summarise the main complexity results of this section.

▶ **Lemma 25.** *Let π be an $(\ell-1)$-length path in the NFA $\mathcal{A}$ ending at q and corresponding to the word w_π, such that $\iota(w_\pi) = c$ and $\mathrm{alph}(\mathrm{r}(w_\pi)) = \mathcal{R}$. Then the word $w_{\pi'}$ corresponding to the path π' formed by following a transition labelled $\mathtt{x} \in \Sigma$ from q either:*

- *has an empty rest ($\mathrm{r}(w_{\pi'}) = \varepsilon$) if $\mathcal{R} \cup \{\mathtt{x}\} = \Sigma$ and hence $\iota(w_{\pi'}) = c+1$, or*
- *has a rest equal to $\mathcal{R} \cup \{\mathtt{x}\}$ ($\mathrm{alph}(\mathrm{r}(w_{\pi'})) = \mathcal{R} \cup \{\mathtt{x}\}$) if $\mathcal{R} \cup \{\mathtt{x}\} \subsetneq \Sigma$ and hence $\iota(w_{\pi'}) = c$.*

Proof. In the first case, if $\mathrm{alph}(\mathrm{r}(w_\pi)) = \Sigma \setminus \{\mathtt{x}\}$ then $\mathrm{r}(w_\pi)\mathtt{x}$ is an arch, as it contains every symbol from Σ at least once, and hence the arch factorisation of $w_{\pi'}$ contains $c+1$ arches, and an empty rest, i.e. $\iota(w_{\pi'}) = c+1$ and $\mathrm{r}(w_{\pi'}) = \varepsilon$. Otherwise, $\mathrm{r}(w_\pi)\mathtt{x}$ contains the set of letters $\mathcal{R} \cup \{\mathtt{x}\}$, and does not complete a new arch. Hence the arch factorisation of $w_{\pi'}$ contains c arches and the rest contains the letters $\mathcal{R} \cup \{\mathtt{x}\}$, i.e., $\iota(w_{\pi'}) = c$ and $\mathrm{alph}(\mathrm{r}(w_{\pi'})) = \mathrm{alph}(w_\pi) \cup \{\mathtt{x}\}$. ◀

Lemma 25 provides the outline of the dynamic programming approach used to compute the table T. Starting with the 0-length path corresponding to the empty word ε, the value of $T[q, \ell, c, \mathcal{R}]$ is computed from the values of $T[q', \ell-1, c', \mathcal{R}']$, allowing an efficient computation of the table. Corollary 26 rewrites this in terms of computing the number of paths of length ℓ ending at state q corresponding to words with c arches and the set of symbols $\mathcal{R}$ of w's rest. The proof is analogous to the one of Lemma 25.

▶ **Corollary 26.** *Let π be an ℓ-length path in the NFA $\mathcal{A}$ with $\iota(w_\pi) = c$, and $\mathrm{alph}(\mathrm{r}(w_\pi)) = \mathcal{R}$. Now, we distinguish the cases whether $\mathcal{R}$ is empty:*
$\mathcal{R} = \emptyset$: $\iota(w_\pi[1, \ell-1]) = c-1$ and $\mathrm{alph}(\mathrm{r}(w_\pi[1, \ell-1])) = \Sigma \setminus \{w_\pi[\ell]\}$,

$\mathcal{R} \neq \emptyset$: $\iota(w_\pi[1, \ell-1]) = c$ *and either* $\mathrm{alph}(\mathrm{r}(w_\pi[1, \ell-1])) = \mathrm{alph}(r(w_\pi)) \setminus \{w_\pi[\ell]\}$ *or* $\mathrm{alph}(\mathrm{r}(w_\pi[1, \ell-1])) = \mathrm{alph}(\mathrm{r}(w_\pi))$. *In this case we also have* $\mathcal{R} = \mathrm{alph}(r(w_\pi[1, \ell-1])) \cup \{w_\pi[\ell]\}$.

Now, we formally establish the dynamic programming approach to calculate the value of $T[q, \ell, c, \mathcal{R}]$ from the already computed cells of the table. For better readability, we use $\mathcal{P}[q, \ell, c, \mathcal{R}]$ as the set of all ℓ-length paths from q_0 to q where the associated word has c arches and the alphabet of its rest is $\mathcal{R}$. Notice that we have $|\mathcal{P}[q, \ell, c, \mathcal{R}]| = T[q, \ell, c, \mathcal{R}]$.

▶ **Lemma 27.** *Let $\mathcal{A}$ be an NFA and assume that $T[q_0, 0, 0, \emptyset]$ is given. Notice that given $q \in Q$, the combination $(q', \mathtt{x}) \in Q \times \Sigma$ only contributes to $T[q, \ell, c, \mathcal{R}]$ if we have $q' \in \Delta(q, \mathtt{x})$. Thus, we have for all $\ell \geq 1$*

$$T[q, \ell, c, \mathcal{R}] = \sum_{\substack{\mathtt{x} \in \Sigma, \\ q' \in \Delta(q, \mathtt{x})}} \begin{cases} 0 & \text{if } \mathtt{x} \notin \mathcal{R} \text{ and } \mathcal{R} \neq \emptyset, \\ 0 & \text{if } \mathcal{R} = \emptyset, c = 0, \\ T[q', \ell-1, c-1, \Sigma \setminus \{\mathtt{x}\}] & \text{if } \mathcal{R} = \emptyset, c > 0, \\ T[q', \ell-1, c, \mathcal{R} \setminus \{\mathtt{x}\}] + T[q', \ell-1, c, \mathcal{R}] & \text{if } \mathcal{R} \neq \emptyset, x \in \mathcal{R}. \end{cases}$$

Proof. From Lemma 25, for every path $\pi \in \mathcal{P}[q, \ell-1, c, \mathcal{R}]$, corresponding to the word w_π, and symbol $\mathtt{x}$, there exists some ℓ-length path π' ending at some state $q' \in \delta(q, x)$ corresponding to the word $w_{\pi'} = w_\pi \mathtt{x}$ such that:

- $\pi' \in \mathcal{P}[q', \ell, c+1, \emptyset]$, if $\mathcal{R} = \Sigma \setminus \{\mathtt{x}\}$, or
- $\pi' \in \mathcal{P}[q', \ell, c, \mathcal{R} \cup \{x\}]$ if $\mathcal{R} \neq \Sigma \setminus \{\mathtt{x}\}$.

In the other direction, from Lemma 26, every path $\pi' \in \mathcal{P}[q, \ell, c, \mathcal{R}]$ corresponding to the word $w_{\pi'}$ must contain an $(\ell-1)$-length prefix corresponding to a word with either:

- $c-1$ arches, and the set of symbols $\Sigma \setminus \{\mathtt{x}\}$ in the rest of the word, if $\mathcal{R} = \emptyset$, or
- c arches, and rest of the word containing the set of symbols $\mathcal{R}$ or $\mathcal{R} \setminus \{\mathtt{x}\}$, if $\mathcal{R} \neq \emptyset$.

Therefore, if $\mathcal{R} = \emptyset$, the size of $\mathcal{P}[q, \ell, c, \mathcal{R}]$ is equal to $\sum_{x \in \Sigma} \sum_{q' \in \Delta(q, \mathtt{x})} T[q', \ell-1, c-1, \Sigma \setminus \{\mathtt{x}\}]$. Similarly, if $\mathcal{R} \neq \emptyset$, the size of $\mathcal{P}[q, \ell, c, \mathcal{R}]$ is equal to $\sum_{x \in \mathcal{R}} \sum_{q' \in \Delta(q, \mathtt{x})} T[q', \ell-1, c, \mathcal{R} \setminus \{x\}] + T[q', \ell-1, c, \mathcal{R}]$.

Note that if $\mathcal{R} \neq \emptyset$, and $\mathtt{x} \notin \mathcal{R}$ for some $\mathtt{x} \in \Sigma$, there is no path in $\mathcal{P}[q, \ell, c, \mathcal{R}]$ where the final transition is labelled $\mathtt{x}$. Similarly , $\mathcal{P}[q, \ell, 0, \emptyset] = \emptyset$ for any $\ell \geq 1$. Otherwise, one of the two above cases must apply, depending on the value of $\mathcal{R}$. This concludes the proof. ◀

Following Lemma 27, the table T can be constructed via dynamic programming. As a base case, note that the only length 0 path in this automaton starting at q_0 is the empty path, which must also end at state q_0 and contains 0 arches and no symbols in the alphabet of the rest. Therefore, $T[q, 0, c, \mathcal{R}]$ is set to 0 for every $q \in Q, \mathcal{R} \subsetneq \Sigma$ and $c \in [0, k-1]$ other than $T[q_0, 0, 0, \emptyset]$, which is set to 1. We now use T to construct U.

▶ **Corollary 28.** *We have* $U[q, \ell] = \sum_{\mathtt{x} \in \Sigma, q' \in \Delta(q, \mathtt{x})} U[q', \ell-1] + T[q', \ell-1, k-1, \Sigma \setminus \{\mathtt{x}\}]$.

Proof. The correctness of this construction follows from Lemma 27. ◀

Note that the total number of k-universal accepting paths of the automaton $\mathcal{A}$ is given by $\sum_{q \in F} U[q, m]$. Using the tables T and U, the total number of k-universal paths and words of length m can be computed in $O^*(m^2 n^2 k 2^\sigma)$.

▶ **Theorem 29.** *The number of k-universal accepting paths of length m of an NFA $\mathcal{A}$, for $k \leq m/\sigma$, can be computed in $O^*(m^2 n^2 k 2^\sigma)$ time. For $k > m/\sigma$, this number is 0.*

Proof. Note that, as explained before, all arithmetic operations require $O(m)$ time in our computational model (we operate with numbers of value at most $(n\sigma)^m$). Using the table U, the number of k-universal words of length m can be counted by computing the sum $\sum_{q\in F} U[q,m]$, which takes $O^*(mn)$ time. To construct the table U, it is nessesary first to construct the table T. Assuming that the value of $T[q',\ell-1,c',\mathcal{R}']$ has been precomputed for every $q'\in Q, c'\in[1,k-1]$ and $\mathcal{R}'\subsetneq\Sigma$, the value of $T[q,\ell,c,\mathcal{R}]$ can be computed in $O^*(mn)$ time. As there are n values of $q\in Q$, 2^σ values of $\mathcal{R}\subset\Sigma$, and k values of $c\in[0,k-1]$, the value of $T[q,\ell,c,\mathcal{R}]$ can be computed for every $q\in Q$, $c\in[0,k-1]$, and $\mathcal{R}\subset\Sigma$ in $O^*(mn^2k2^\sigma)$ time. As there are $m+1$ values of $\ell\in[0,m]$, the total time of constructing T is $O^*(m^2n^2k2^\sigma)$.

Analogously, the value of $U[q,\ell]$ can be computed in $O^*(mn)$ time, assuming the values of $U[q',\ell-1]$ and $T[q',\ell-1,k-1,\Sigma\setminus\{\mathtt{x}\}]$ have be precomputed from every $q'\in Q$ and $\mathtt{x}\in\Sigma$. Hence U can be constructed from the table T in $O^*(m^2n^2)$ time. By extension the number of k-universal paths of length m accepted by $\mathcal{A}$ computed in $O^*(m^2n^2k2^\sigma)$ time. ◀

▶ **Corollary 30.** *The number of k-universal accepting paths of length at most m of an NFA $\mathcal{A}$, for $k\le m/\sigma$, can be computed in $O^*(m^2n^2k2^\sigma)$ time. For $k>m/\sigma$, this number is* 0*.*

Proof. Observe that the number of k-universal paths of length at most ℓ is equal to $\sum_{q\in F}\sum_{\ell\in[1,m]} U[q,\ell]$. As this summation can be computed in $O^*(mn)$ time once U has been constructed, the total complexity follows from the cost of constructing tables T and U. ◀

Since in a DFA each word is associated with exactly one path, the following holds.

▶ **Corollary 31.** *The number of k-universal words of length either exactly or at most m accepted by a DFA $\mathcal{A}$, for $k\le m/\sigma$, can be computed in $O^*(m^2n^2k2^\sigma)$ time. For $k>m/\sigma$, this number is* 0*.*

We may further generalise this to the problem of finding the number of perfect k-universal words and paths by discarding any k-universal paths with a non-empty rest.

▶ **Corollary 32.** *The number of perfect k-universal accepting paths of length either exactly or at most m of an NFA $\mathcal{A}$, for $k\le m/\sigma$, can be computed in $O^*(m^2n^2k2^\sigma)$ time.*

▶ **Corollary 33.** *The number of perfect k-universal words of length either exactly or at most m accepted by a DFA $\mathcal{A}$, for $k\le m/\sigma$, can be computed in $O^*(m^2n^2k2^\sigma)$ time.*

We now generalise these tools to the problem of counting the total number of k-universal paths accepted by a finite automaton $\mathcal{A}$. The primary challenge of counting the total number of such paths comes from determining if there exists either a finite or an infinite number of k-universal paths accepted by $\mathcal{A}$. By the pumping lemma [26], we have that an automaton $\mathcal{A}$ accepts an infinite number of k-universal words (paths) if and only if $\mathcal{A}$ accepts some k-universal word (path) of length at least $n+1$. Clearly, if $k>n$ we have that $\mathcal{A}$ either accepts an infinite number of k-universal words (paths) or none (and this can be tested as in Lemma 19, in time that does not depend on k). Therefore, using the upper bound on the maximum length of the shortest k-universal word accepted by the automaton $\mathcal{A}$ from Lemma 18, combined with Corollary 31, we now count the total number of k-subsequence universal words accepted by $\mathcal{A}$.

▶ **Theorem 34.** *The total number of k-universal words (resp., paths) accepted by a DFA (resp., NFA) $\mathcal{A}$ can be determined in $O^*(n^4k^22^\sigma)$ time (resp., $O^*(n^4k^32^\sigma)$ time), for $k\le n$. For $k>n$, this number is either* 0 *or* ∞*, and can be determined in $O^*(n^32^\sigma)$ time.*

Proof. We only show this for NFAs, as the argument for DFAs is similar (and uses the previous results corresponding to this class of automata).

Note first that if an automaton accepts some k-universal paths w of length at least $n+1$, then it must accept an infinite number of such words, as the path induced by w in $\mathcal{A}$ must visit some state twice and thus contain a cycle. Therefore, if $\mathcal{A}$ accepts only a finite number of k-universal paths, the total number of paths accepted by $\mathcal{A}$ can be computed in $O^*(n^4k2^\sigma)$ time via Theorem 29 and Corollary 30.

Following the same arguments as in Lemma 18, a k-universal path of length of at least $n+1$ exists if and only if there exists some k-universal word of length between $n+1$ and $kn\sigma$, hence it is sufficient to check if $\mathcal{A}$ accepts some word of length between $n+1$ and $kn\sigma$, which can be achieved in $O^*(n^4k^32^\sigma)$ time. If no k-universal word accepted by $\mathcal{A}$ of length at least $n+1$ exists, then total number of k-universal words accepted by $\mathcal{A}$ is determined by counting the number of k-universal words accepted by $\mathcal{A}$ of length at most n. ◀

We now extend our results of counting to the problem of ranking k-universal words. Note, that these results can be generalised to ranking the k-universal accepted paths, but one needs to define an ordering on the transitions from each state in the automaton. To keep the presentation simple, we will therefore only discuss here about DFAs and words. The main idea is to count the number of k-universal words with a prefix strictly smaller than the prefix of w of the same length; again, each arithmetic operation takes $O(m)$ time in our computational model. This section is laid out as follows. First, we show how to compute the rank of w efficiently within the set $\mathrm{Univ}_{\mathcal{A}_{\leq m},k}$ in $O^*(m^2n^2k2^\sigma)$ time. Secondly, we show that the rank of w can be computed within the set $\mathrm{Univ}_{L(\mathcal{A}),k}$ in $O^*(n^4k^32^\sigma)$ time. As noted above, when discussing counting, these problems only make sense for $k \leq m/\sigma$. This follows from the same arguments used to count the total number of k-universal words accepted by $\mathcal{A}$ laid out in Theorem 34.

The primary tool used in this section is a generalisation of the path table: the *fixed prefix path table of length* m is an $n \times (m+1) \times k \times 2^\sigma$ sized table, defined for a set of prefixes $\mathcal{PR}$ and denoted $T(\mathcal{PR})$. Informally, the table $T(\mathcal{PR})$ is used to count the number of paths with some prefix from the given set $\mathcal{PR}$. Thus, $T(\mathcal{PR})[q,\ell,c,\mathcal{R}]$ stores the number of words $w \in \Sigma^\ell$ associated to a path from q_0 to q, with $\iota(w) = c$, $\mathrm{alph}(r(w)) = \mathcal{R}$, and there exists a $p \in \mathcal{PR}$ such that $p = w[1,|p|]$. The table $U(\mathcal{PR})$ is defined analogously to the table U but again with the additional condition as in $T(\mathcal{PR})$. These tables can be constructed directly using the same techniques introduced for T and U, by initially setting $T(\mathcal{PR})[q_0,0,0,\emptyset]$ to 0 and $T(\mathcal{PR})[\delta(q_0,p),|p|,\iota(p),\mathrm{alph(r(}p\mathrm{))}]$ to 1 for every $p \in \mathcal{PR}$. Similarly, in the special case where there exists some $p \in \mathcal{PR}$ such that $\iota(p) \geq k$, then the value of $U[\delta(q_0,p),|p|]$ is set to 1. We assume, without loss of generality, that no prefix in $\mathcal{PR}$ is also the prefix of some other word $p' \in \mathcal{PR}$. The remaining entries are computed as before. In the following results, we use $\mathcal{PR}(w) = \{w[1,i]\mathtt{x} \mid i \in [0,|w|], \mathtt{x} \in [1, w[i+1]-1]\}$. Note, that the following results hold for both counting words accepted by deterministic automata, and accepting paths in non-deterministic automata.

▶ **Corollary 35.** *$T(\mathcal{PR})$, $U(\mathcal{PR})$ are constructible for m-length paths in $O^*(m^2n^2k2^\sigma)$ time.*

▶ **Theorem 36.** *The rank of $w \in \mathrm{Univ}_{\mathcal{A}_m,k}$ can be determined in $O^*(m^2n^2k2^\sigma)$ time.*

Proof. Note that a word v is smaller than w (w.r.t. the lexicographical ordering) if and only if v is a prefix of w or they share a common prefix u and $v[|u|+1] < w[|u|+1]$. Therefore, the number of m-length k-universal words starting with some prefix in $\mathcal{PR}(w)$ is given by either $\sum_{q\in F} U(\mathcal{PR}(w))[q,m]$, if w has length at most m, or $1 + \sum_{q\in F} U(\mathcal{PR}(w))[q,m]$

if the $w[1,m]^{\text{th}}$ state of the path associated with w is an accepting state, $|w| > m$, and $\iota(w[1,m]) = k$. As $T(\mathcal{PR}(w))$ can be computed in $O^*(m^2n^2k2^\sigma)$ time, and the above summation completed in $O^*(mn)$ time, the total time complexity of finding the m-length rank of w is $O^*(m^2n^2k2^\sigma)$. ◀

▶ **Corollary 37.** *The rank of $w \in \mathrm{Univ}_{\mathcal{A}_{\leq m},k}$ can be determined in $O^*(m^2n^2k2^\sigma)$ time.*

Proof. Using the table $T(\mathcal{PR}(w))$ as above, the number of words k-universal words of length at most m smaller than w is given by

$$\sum_{i\in[1,m]}\sum_{q\in F} U(\mathcal{PR}(w))[q,i] + \sum_{i\in[1,m]} \begin{cases} 1, & q_{w[1,i]} \in F, \\ 0, & q_{w[1,i]} \notin F. \end{cases}$$

As the table can be constructed in $O^*(m^2n^2k2^\sigma)$ time, and the summation requires at most $O^*(m^2n)$ time, the total complexity of finding the at-most-m-length rank of a word w in $\mathrm{Univ}_{\mathcal{A}_m,k}$ is $O^*(m^2n^2k2^\sigma)$. ◀

▶ **Corollary 38.** *The rank of $w \in \mathrm{Univ}_{L(\mathcal{A}),k}$ can be determined in $O^*(n^4k^32^\sigma)$ time.*

Proof. Following the same arguments as given in Theorem 34, note that there is an infinite number of words smaller than w if and only if there exists some word of length at least $n+1$ with a prefix in $\mathcal{PR}$. The existence of such a word can be determined from the tables $T(\mathcal{PR}(w))$ and $U(\mathcal{PR}(w))$ for paths of length at most $km\sigma$ in $O^*(k^2n^3)$ time. As the tables $T(\mathcal{PR}(w))$ and $U(\mathcal{PR}(w))$ can be constructed for paths of length at most $kn\sigma$ in $O^*(n^4k^32^\sigma)$ time, the total rank of w within $\mathrm{Univ}_{L(\mathcal{A}),k}$ can be computed in $O^*(n^4k^2)$ time. ◀

5 Conclusions

This paper proposed a series of novel algorithmic results and insights regarding the analysis of the sets which can be expressed as the intersection of regular languages and the language of k-universal words over some alphabet. We have introduced two natural notions of k-universality in regular languages, namely existence k-universal languages and universal k-universal languages, and have proposed algorithms for testing whether a regular language is in one of these two classes. While we have a good understanding of the problem of deciding whether a language is defined by an existence k-universal automaton, the exact complexity of the problem of deciding whether a language is universal k-universal remains open. As well as the introduction of these notions, and the study of some decisions problems related to them, we have provided a toolbox for counting and ranking k-universal paths (respectively, words) accepted by a given NFA (respectively, DFA).

We note that using a divide and conquer approach to count paths (with a certain amount of arches, at most m) of length 2^ℓ by combining paths of length $2^{\ell-1}$ (with less arches), one factor m can be reduced to $\log m$ for counting and ranking words of (or of at most) given length m, at the cost of additional complexity in terms of n, k and σ (as, in that case, one would have to allow the existence of prefixes of such paths which are not part of arches, as well as consider the fact that these paths connect arbitrary pairs of states, and may have different counts of arches). This has been left out of the current version to provide a clearer understanding of our main results, and avoid over-complicating the presentation of the paper.

Duncan Adamson's work was funded by the Leverhulme Trust via the Leverhulme Research Centre for Functional Material Design. Tore Koß's work was supported by the DFG project number 389613931. Florin Manea's work was supported by the DFG Heisenberg-project number 466789228.

References

1 D. Adamson. Ranking binary unlabelled necklaces in polynomial time. In *DCFS*, pages 15–29. Springer, 2022.

2 D. Adamson. Ranking and unranking k-subsequence universal words. In Anna Frid and Robert Mercaş, editors, *WORDS*, pages 47–59. Springer Nature Switzerland, 2023.

3 D. Adamson, A. Deligkas, V. V. Gusev, and I. Potapov. Ranking bracelets in polynomial time. *CPM*, pages 4–17, 2021.

4 D. Adamson, M. Kosche, T. Koß, F. Manea, and S. Siemer. Longest common subsequence with gap constraints. In Anna Frid and Robert Mercaş, editors, *WORDS*, pages 60–76, 2023.

5 A. Artikis, A. Margara, M. Ugarte, S. Vansummeren, and M. Weidlich. Complex event recognition languages: Tutorial. In *DEBS*, pages 7–10, 2017.

6 L. Barker, P. Fleischmann, K. Harwardt, F. Manea, and D. Nowotka. Scattered factor-universality of words. In *DLT*, pages 14–28. Springer, 2020.

7 H. Z. Q. Chen, S. Kitaev, T. Mütze, and B. Y. Sun. On universal partial words. *Electronic Notes in Discrete Mathematics*, 61:231–237, 2017.

8 M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

9 J.D. Day, P. Fleischmann, M. Kosche, T. Koß, F. Manea, and S. Siemer. The edit distance to k-subsequence universality. In *STACS*, volume 187, pages 25:1–25:19, 2021.

10 N. G. de Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 49:758–764, 1946.

11 L. Fleischer and M. Kufleitner. Testing simon's congruence. In *MFCS*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

12 P. Fleischmann, S.B. Germann, and D. Nowotka. Scattered factor universality–the power of the remainder. *preprint arXiv:2104.09063 (published at RuFiDim)*, 2021.

13 P. Fleischmann, L. Haschke, A. Huch, A. Mayrock, and D. Nowotka. Nearly k-universal words-investigating a part of simon's congruence. In *DCFS*, pages 57–71, 2022.

14 P. Fleischmann, J. Höfer, A. Huch, and D. Nowotka. α-β-factorization and the binary case of simon's congruence, 2023. `arXiv:2306.14192`.

15 F. V. Fomin, D. Kratsch, I. Todinca, and Y. Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM J. Comput.*, 38(3):1058–1079, 2008. `doi:10.1137/050643350`.

16 H. Fredricksen and J. Maiorana. Necklaces of beads in k colors and k-ary de Bruijn sequences. *Discrete Mathematics*, 23(3):207–210, 1978.

17 A. Frochaux and S. Kleest-Meißner. Puzzling over subsequence-query extensions: Disjunction and generalised gaps. In *AMW 2023*, volume 3409 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

18 P. Gawrychowski, M. Kosche, T. Koß, F. Manea, and S. Siemer. Efficiently Testing Simon's Congruence. In *STACS*, volume 187, pages 34:1–34:18, 2021.

19 P. Gawrychowski, M. Lange, N. Rampersad, J. O. Shallit, and M. Szykula. Existential length universality. In *Proc. STACS 2020*, volume 154 of *LIPIcs*, pages 16:1–16:14, 2020.

20 E. N. Gilbert and J. Riordan. Symmetry types of periodic sequences. *Illinois Journal of Mathematics*, 5(4):657–665, 1961.

21 B. Goeckner, C. Groothuis, C. Hettle, B. Kell, P. Kirkpatrick, R. Kirsch, and R. W. Solava. Universal partial words over non-binary alphabets. *Theor. Comput. Sci*, 713:56–65, 2018.

22 S. Halfon, P. Schnoebelen, and G. Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *LICS*, pages 1–12. IEEE, 2017.

23 R. Han, S. Wang, and X. Gao. Novel algorithms for efficient subsequence searching and mapping in nanopore raw signals towards targeted sequencing. *Bioinformatics*, 36(5):1333–1343, 2020.

24 J.-J. Hebrard. An algorithm for distinguishing efficiently bit-strings by their subsequences. *Theoretical Computer Science*, 82(1):35–49, 1991.

25 M. Holzer and M. Kutrib. Descriptional and computational complexity of finite automata - A survey. *Inf. Comput.*, 209(3):456–470, 2011.

26 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

27 P. Karandikar, M. Kufleitner, and P. Schnoebelen. On the index of Simon's congruence for piecewise testability. *Inf. Process. Lett.*, 115(4):515–519, 2015.

28 P. Karandikar and P. Schnoebelen. The height of piecewise-testable languages with applications in logical complexity. In *CSL*, 2016.

29 S. Kim, Y. Han, S. Ko, and K. Salomaa. On simon's congruence closure of a string. In *DCFS 2022, Proceedings*, volume 13439 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 2022.

30 S. Kim, Y. Han, S. Ko, and K. Salomaa. On the simon's congruence neighborhood of languages. In *DLT 2023, Proceedings*, volume 13911 of *Lecture Notes in Computer Science*, pages 168–181. Springer, 2023.

31 S. Kim, S. Ko, and Y. Han. Simon's congruence pattern matching. In *ISAAC 2022, Proceedings*, volume 248 of *LIPIcs*, pages 60:1–60:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

32 S. Kleest-Meißner, R. Sattler, M. L. Schmid, N. Schweikardt, and M. Weidlich. Discovering event queries from traces: Laying foundations for subsequence-queries with wildcards and gap-size constraints. In *ICDT 2022, Proceedings*, volume 220 of *LIPIcs*, pages 18:1–18:21, 2022.

33 S. Kleest-Meißner, R. Sattler, M. L. Schmid, N. Schweikardt, and M. Weidlich. Discovering multi-dimensional subsequence queries from traces - from theory to practice. In *BTW 2023, Proceedings*, volume P-331 of *LNI*, pages 511–533, 2023.

34 T. Kociumaka, J. Radoszewski, and W. Rytter. Computing k-th Lyndon word and decoding lexicographically minimal de Bruijn sequence. In *CPM*, pages 202–211. Springer, 2014.

35 M. Kosche, T. Koß, F. Manea, and S. Siemer. Absent subsequences in words. In *RP*, pages 115–131. Springer, 2021.

36 M. Kosche, T. Koß, F. Manea, and S. Siemer. Combinatorial algorithms for subsequence matching: A survey. In Henning Bordihn, Géza Horváth, and György Vaszil, editors, *NCMA*, 2022.

37 M. Krötzsch, T. Masopust, and M. Thomazo. Complexity of universality and related problems for partially ordered NFAs. *Inf. Comput.*, 255:177–192, 2017.

38 M. Lothaire. *Combinatorics on Words*. Cambridge Mathematical Library. Cambridge University Press, 1997.

39 M. H. Martin. A problem in arrangements. *Bull. Amer. Math. Soc.*, 40(12):859–864, December 1934.

40 A. Mateescu, A. Salomaa, and S. Yu. Subword histories and parikh matrices. *Journal of Computer and System Sciences*, 68(1):1–21, 2004.

41 N. Rampersad, J. Shallit, and Z. Xu. The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. *Fundam. Inf.*, 116(1-4):223–236, January 2012.

42 C. Savage. A survey of combinatorial gray codes. *SIAM review*, 39(4):605–629, 1997.

43 J. Sawada and A. Williams. Practical algorithms to rank necklaces, Lyndon words, and de Bruijn sequences. *Journal of Discrete Algorithms*, 43:95–110, 2017.

44 P. Schnoebelen and P. Karandikar. The height of piecewise-testable languages and the complexity of the logic of subwords. *Logical Methods in Computer Science*, 15, 2019.

45 P. Schnoebelen and J. Veron. On arch factorization and subword universality for words and compressed words. In *WORDS 2023, Proceedings*, volume 13899 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 2023.

46 A. C. Shaw. Software descriptions with flow expressions. *IEEE Transactions on Software Engineering*, 3:242–254, 1978.

47 R. Shikder, P. Thulasiraman, P. Irani, and P. Hu. An openmp-based tool for finding longest common subsequence in bioinformatics. *BMC research notes*, 12:1–6, 2019.

48 I. Simon. Piecewise testable events. In *Autom. Theor. Form. Lang., 2nd GI Conf.*, volume 33 of *LNCS*, pages 214–222. Springer, 1975.

49 I. Simon. Words distinguished by their subwords. *WORDS*, 27:6–13, 2003.

50 Z. Troniĉek. Common subsequence automaton. In *CIAA*, pages 270–275, 2003.

51 G. Zetzsche. The complexity of downward closure comparisons. In *ICALP*, volume 55, pages 123:1–123:14, 2016.

Unified Almost Linear Kernels for Generalized Covering and Packing Problems on Nowhere Dense Classes

Jungho Ahn
Korea Institute for Advanced Study, Seoul, South Korea

Jinha Kim
Department of Mathematics, Chonnam National University, Gwangju, South Korea

O-joung Kwon
Department of Mathematics, Hanyang University, Seoul, South Korea
Discrete Mathematics Group, Institute for Basic Science, Daejeon, South Korea

Abstract

Let $\mathcal{F}$ be a family of graphs, and let p, r be nonnegative integers. For a graph G and an integer k, the $(p, r, \mathcal{F})$-Covering problem asks whether there is a set $D \subseteq V(G)$ of size at most k such that if the p-th power of G has an induced subgraph isomorphic to a graph in $\mathcal{F}$, then it is at distance at most r from D. The $(p, r, \mathcal{F})$-Packing problem asks whether G^p has k induced subgraphs $H_1, \ldots, H_k$ such that each H_i is isomorphic to a graph in $\mathcal{F}$, and for $i, j \in \{1, \ldots, k\}$, the distance between $V(H_i)$ and $V(H_j)$ in G is larger than r.

We show that for every fixed nonnegative integers p, r and every fixed nonempty finite family $\mathcal{F}$ of connected graphs, $(p, r, \mathcal{F})$-Covering with $p \leqslant 2r + 1$ and $(p, r, \mathcal{F})$-Packing with $p \leqslant 2\lfloor r/2 \rfloor + 1$ admit almost linear kernels on every nowhere dense class of graphs, parameterized by the solution size k. As corollaries, we prove that Distance-r Vertex Cover, Distance-r Matching, $\mathcal{F}$-Free Vertex Deletion, and Induced-$\mathcal{F}$-Packing for any fixed finite family $\mathcal{F}$ of connected graphs admit almost linear kernels on every nowhere dense class of graphs. Our results extend the results for Distance-r Dominating Set by Drange et al. (STACS 2016) and Eickmeyer et al. (ICALP 2017), and for Distance-r Independent Set by Pilipczuk and Siebertz (EJC 2021).

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases kernelization, independent set, dominating set, covering, packing

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.5

Related Version *Full Version*: `https://arxiv.org/abs/2207.06660`

Funding All the authors were supported by the Institute for Basic Science (IBS-R029-C1). Jungho Ahn was also supported by the KIAS Individual Grant (CG095301) at Korea Institute for Advanced Study, and O-joung Kwon was also supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (No. NRF-2021K2A9A2A11101617 and RS-2023-00211670).

1 Introduction

The Dominating Set problem is one of the classical NP-hard problems which asks whether a graph G contains a set of at most k vertices whose closed neighborhood contains all the vertices of G. A natural variant of it is the Distance-r Dominating Set problem which asks whether G contains a set of at most k vertices such that every vertex of G is at distance at most r from one of these vertices. Dominating Set has been intensively studied in the context of fixed-parameter algorithms. In a parameterized problem Π, we are given an

instance (x, k) where k is a parameter, and the central question is whether the parameterized problem admits an algorithm, called *fixed-parameter* algorithm, that runs in time $f(k) \cdot |x|^c$ for some computable function f and a constant c. We say that Π is *fixed-parameter tractable*, or FPT for short if it admits a fixed-parameter algorithm. It is known that Dominating Set is W[2]-complete parameterized by k [14, 15], meaning that it is not FPT unless an unexpected collapse occurs in the parameterized complexity hierarchy. Thus, it is natural to restrict graph classes and see whether a fixed-parameter algorithm exists. Dominating Set admits a fixed-parameter algorithm on planar graphs [13, 27], and the project of finding larger sparse graph classes on which fixed-parameter algorithms for Dominating Set exist has been studied intensively, see [11, 4, 30, 38, 29, 24, 40].

A *kernelization* algorithm for a parameterized problem takes an instance (x, k) and outputs an equivalent instance (x', k') in time polynomial in $|x| + k$, where $|x'| + k' \leqslant g(k)$ for some computable function g. We call the function g the *size* of the kernel. If g is a polynomial (resp. linear), then such an algorithm is called a polynomial (resp. linear) kernel. It is well known that a parameterized decision problem is FPT if and only if it admits a kernelization; see [16]. Furthermore, with a polynomial kernel, we can compress inputs to instances of polynomial size, which lead to boost up the running time of exact algorithms solving the problem, like the brute-force search algorithm. Therefore, it is natural and applicable to investigate the existence of a polynomial kernel or a linear kernel. In particular, the existence of linear kernels for Dominating Set on sparse graph classes have been investigated.

One of the first results is a linear kernel for Dominating Set on planar graphs due to Alber, Fellows, and Niedermeier [3]. It has been generalized to classes of bounded genus graphs [26], H-minor free graphs [23], and H-topological minor free graphs [24]. Drange et al. [17] extended the previous results to classes of graphs with bounded expansion for Distance-r Dominating Set, and Eickmeyer et al. [20] obtained almost linear kernels for Distance-r Dominating Set on nowhere dense classes of graphs. Classes of graphs with bounded expansion and nowhere dense classes of graphs were introduced by Nešetřil and Ossona de Mendez [37], which are defined in terms of shallow minors and capture most of well-studied sparse graph classes.

Independent Set is another classic NP-hard problem which asks to find a set of k vertices in a given graph whose pairwise distance is more than 1, and Distance-r Independent Set is the problem obtained by replacing 1 with r. It is known that Independent Set is W[1]-complete parameterized by k [15]. The distance variations of Dominating Set and Independent Set are closely related, in a sense that the size of a distance-$2r$ independent set is a lower bound for the minimum size of a distance-r dominating set. Dvořák [18] presented an approximation algorithm for Distance-r Dominating Set, which outputs a set of size bounded by a function of the $2r$-weak coloring number and the maximum size of a distance-$2r$ independent set. Pilipczuk and Siebertz [39] recently presented an almost linear kernel for Distance-r Independent Set on nowhere dense classes of graphs.

For a fixed r, both Distance-r Dominating Set and Distance-r Independent Set can be expressed in first-order logic. Thus, by the meta-theorem of Grohe, Kreutzer, and Siebertz [29], there are almost-linear-time fixed-parameter tractable on every nowhere dense class of graphs. Fabiański, Pilipczuk, Siebertz, and Toruńczyk [22] presented linear-time fixed-parameter algorithms for Distance-r Dominating Set on various graph classes, including powers of nowhere dense classes and map graphs, and a linear-time fixed-parameter algorithm for Distance-r Independent Set on every nowhere dense class of graphs.

A natural question is whether there are linear/polynomial kernels for other problems on classes of graphs with bounded expansion and nowhere dense classes of graphs. Meta-type kernelization results have been studied for graphs on bounded genus [6], H-minor free

graphs [25], H-topological minor free graphs [32], classes of graphs with bounded expansion, and nowhere dense classes of graphs [28]. Note that the last result by Gajarsky et al. [28] is to obtain kernelizations parameterized by the size of a modulator to constant tree-depth, and not by the solution size. Currently, limited investigations have been conducted on variations of Distance-r Dominating Set and Distance-r Independent Set. In this paper, we consider generic problems to hit finite graphs by the r-th neighborhood of a set of vertices.

Let $\mathcal{F}$ be a family of graphs, and let p and r be nonnegative integers. For a graph G, let G^p be the graph with vertex set $V(G)$ such that distinct vertices v and w are adjacent in G^p if and only if the distance between v and w in G is at most p, and let $N_G^r[D]$ be the set of all vertices in G at distance at most r from D in G. For a graph G, a set $D \subseteq V(G)$ is a $(p, r, \mathcal{F})$-*cover* of G if there is no set $X \subseteq V(G) \setminus N_G^r[D]$ such that $G^p[X]$ is isomorphic to a graph in $\mathcal{F}$. We denote by the $\gamma_{p,r}^{\mathcal{F}}(G)$ the minimum size of a $(p, r, \mathcal{F})$-cover of G. For a graph G and an integer k, the $(p, r, \mathcal{F})$-Covering problem asks whether $\gamma_{p,r}^{\mathcal{F}}(G) \leqslant k$. Note that Distance-$r$ Dominating Set is equal to $(1, r, \{K_1\})$-Covering.

Our main results are the following. For classes of graphs with bounded expansion, we can obtain linear kernels. Let $\mathbb{N}$ be the set of nonnegative integers and $\mathbb{R}_+$ be the set of positives.

▶ **Theorem 1.1.** *For every nowhere dense class $\mathcal{C}$ of graphs, there is $g_{\mathrm{cov}} : \mathbb{N} \times \mathbb{N} \times \mathbb{R}_+ \to \mathbb{N}$ such that for every nonempty family $\mathcal{F}$ of connected graphs with at most d vertices, $p, r \in \mathbb{N}$ with $p \leqslant 2r + 1$, and $\varepsilon > 0$, there is a polynomial-time algorithm that given a graph $G \in \mathcal{C}$ and $k \in \mathbb{N}$, either correctly decides that $\gamma_{p,r}^{\mathcal{F}}(G) > k$, or outputs a graph G' with $|V(G')| \leqslant g_{\mathrm{cov}}(r, d, \varepsilon) \cdot k^{1+\varepsilon}$ such that $\gamma_{p,r}^{\mathcal{F}}(G) \leqslant k$ if and only if $\gamma_{p,r}^{\mathcal{F}}(G') \leqslant k + 1$.*

A $(p, r, \mathcal{F})$-*packing* of G is a family of sets $A_1, \ldots, A_\ell \subseteq V(G)$ such that each $G^p[A_i]$ is isomorphic to a graph in $\mathcal{F}$, and for all $1 \leqslant i < j \leqslant \ell$, the distance between A_i and A_j in G is more than r. We denote by $\alpha_{p,r}^{\mathcal{F}}(G)$ the maximum size of a $(p, r, \mathcal{F})$-packing of G. For a graph G and an integer k, the $(p, r, \mathcal{F})$-Packing problem asks whether $\alpha_{p,r}^{\mathcal{F}}(G) \geqslant k$. Note that Distance-$r$ Independent Set is equal to $(1, r, \{K_1\})$-Packing.

▶ **Theorem 1.2.** *For every nowhere dense class $\mathcal{C}$ of graphs, there is $g_{\mathrm{pack}} : \mathbb{N} \times \mathbb{N} \times \mathbb{R}_+ \to \mathbb{N}$ such that for every nonempty family $\mathcal{F}$ of connected graphs with at most d vertices, $p, r \in \mathbb{N}$ with $p \leqslant 2\lfloor r/2 \rfloor + 1$, and $\varepsilon > 0$, there is a polynomial-time algorithm that given a graph $G \in \mathcal{C}$ and $k \in \mathbb{N}$, either correctly decides that $\alpha_{p,r}^{\mathcal{F}}(G) = 0$, or correctly decides that $\alpha_{p,r}^{\mathcal{F}}(G) > k$, or outputs a graph G' with $|V(G')| \leqslant g_{\mathrm{pack}}(r, d, \varepsilon) \cdot k^{1+\varepsilon}$ such that $\alpha_{p,r}^{\mathcal{F}}(G) \geqslant k$ if and only if $\alpha_{p,r}^{\mathcal{F}}(G') \geqslant k + 1$.*

Applications. Our kernels for the covering problems have the following applications. Canales, Hernández, Martins, and Matos [9] introduced a distance-r vertex cover and a distance-r guarding set. For a graph G and a positive integer r, a set $D \subseteq V(G)$ is a *distance-r vertex cover* if $G - N_G^r[D]$ has no edge, and a *distance-r guarding set* if $G - N_G^{r-1}[D]$ has no triangle. For a positive integer k, the Distance-r Vertex Cover problem asks whether a graph G has a distance-r vertex cover of size at most k. Similarly, the Distance-r Guarding Set problem asks whether a graph G has a distance-r guarding set of size at most k. Distance-r Vertex Cover and Distance-r Guarding Set for $r \geqslant 1$ can be formulated as $(1, r, \{K_2\})$-Covering and $(1, r - 1, \{K_3\})$-Covering, respectively. By Theorem 1.1, both problems admit almost linear kernels on every nowhere dense class of graphs, and these kernels also can be translated to fixed-parameter algorithms solving those problems on every nowhere dense class of graphs.

For a family $\mathcal{F}$ of graphs, a graph G, and an integer k, the $\mathcal{F}$-Free Vertex Deletion problem asks whether there is a set S of at most k vertices in G such that $G - S$ has no induced subgraph isomorphic to $\mathcal{F}$. If all graphs in $\mathcal{F}$ have at most d vertices, then this problem is related to the d-Hitting Set problem, and has a kernel of size $\mathcal{O}(k^{d-1})$ [1]. Our results imply that if $\mathcal{F}$ is a finite set of connected graphs, then $\mathcal{F}$-Free Vertex Deletion admits an almost linear kernel on every nowhere dense class of graphs. This can be applied to Cograph Vertex Deletion [36], to Cluster Vertex Deletion [5, 41], and to Claw-Free Vertex Deletion [7].

Our kernels for the packing problems have the following applications. A matching of a graph G is a set M of edges of G such that no two edges in M share an end. For a positive integer r, a *distance-r matching* of G is a matching M of G such that for distinct edges $u_1u_2, v_1v_2 \in M$, $\min\{\mathrm{dist}_G(u_i, v_j) : i, j \in [2]\} \geqslant r$. For an integer k, the Distance-r Matching problem asks whether G has a distance-r matching of size at least k. Distance-1 Matching is nothing but finding a matching of size at least k, so can be solved in polynomial time [19]. Distance-r Matching for $r \geqslant 2$ is equal to $(1, r-1, \{K_2\})$-Packing. Moser and Sikdar [35] presented linear kernels for Distance-2 Matching on planar graphs and graphs of bounded degree, and a cubic kernel for the same problem on graphs of girth at least 6. Later, Kanj, Pelsmajer, Schaefer, and Xia [31] presented a kernel of size $40k$ for Distance-2 Matching on planar graphs. By Theorem 1.2, Distance-r Matching for every $r \geqslant 2$ admits an almost linear kernel on every nowhere dense class of graphs.

We can further generalize the matching problem. For a graph H and an integer k, the H-Matching problem asks whether a graph G has k vertex-disjoint subgraphs isomorphic to H. For every integer $d \geqslant 1$, let P_d be a path on d vertices. Dell and Marx [10] presented a kernel for P_3-Matching with $O(k^{2.5})$ edges, and a unified kernel for P_d-Matching with $O(d^{d^2}d^7k^3)$ vertices. They also showed that for every integer $d \geqslant 3$ and every $\varepsilon > 0$, under some complexity hypothesis, K_d-Matching does not have kernels with $O(k^{d-1-\varepsilon})$ edges. By taking $\mathcal{F}$ as the set of all graphs on $|V(H)|$ vertices that contain H as a subgraph, we can formulate H-Matching as $(1, 0, \mathcal{F})$-Packing. Generally, we may consider Induced-$\mathcal{F}$-Packing which asks whether a graph has k vertex-disjoint induced subgraphs each isomorphic to some graph in $\mathcal{F}$. By Theorem 1.2, Induced-$\mathcal{F}$-Packing for every fixed finite family $\mathcal{F}$ of connected graphs admits almost linear kernel on every nowhere dense class of graphs.

We may formulate $(p, r, \mathcal{F})$-Covering and $(p, r, \mathcal{F})$-Packing on fixed powers of a given graph. Formally, for a fixed positive integer t, $(pt, rt, \mathcal{F})$-Covering on a graph G is exactly same as $(p, r, \mathcal{F})$-Covering on its t-th power G^t. Therefore, our result provides the existence of almost linear kernels for both problems on t-th powers G^t of graphs from a nowhere dense class of graphs, assuming that the original graph G is given. However, if the power G^t is only given, then we need to find the graph G to apply our kernelization algorithm.

Organization. We organize this paper as follows. In Section 2, we present some terminology from graph theory, especially lemmas on nowhere dense classes of graphs. In Sections 3 and 4, we present almost linear kernels for $(p, r, \mathcal{F})$-Covering and $(p, r, \mathcal{F})$-Packing on every nowhere dense class of graphs, respectively.

2 Preliminaries

In this paper, all graphs are simple and finite and have at least one vertex. For an equivalence relation $\sim$ on a set X, we denote by $\mathrm{index}(\sim)$ the number of equivalence classes of $\sim$ in X. For every integer n, let $[n]$ be the set of positive integers at most n. Throughout this section,

we let p, r be nonnegative integers, let G be a graph, and let A, B be subsets of $V(G)$. We follow the notations from the textbook of Diestel [12]. We denote by $\mathrm{dist}_G(v, w)$ the distance between vertices v and w in G and by $\mathrm{dist}_G(A, B)$ be the shortest distance between a vertex in A and a vertex in B. The *p-th power* of G, denoted by G^p, is the graph with vertex set $V(G)$ such that distinct vertices v and w are adjacent in G^p if and only if $\mathrm{dist}_G(v, w) \leqslant p$. For a vertex v of G, let $N_G^r[v]$ be the set of vertices of G which are at distance at most r from v in G, and $N_G^r(v) := N_G^r[v] \backslash \{v\}$. Let $N_G^r[A] := \bigcup_{v \in A} N_G^r[v]$ and $N_G^r(A) := N_G^r[A] \backslash A$.

A set $X \subseteq V(G)$ is a *distance-r independent set* in G if the vertices in X are pairwise at distance larger than r in G. We denote by $\alpha_r(G)$ the maximum size of a distance-r independent in G. A set $D \subseteq V(G)$ is a *distance-r dominating set* of G if every vertex of G lies in $N_G^r[D]$. We denote by $\gamma_r(G)$ the minimum size of a distance-r dominating set of G.

Sparse graphs. A graph H with vertex set $\{v_1, \ldots, v_n\}$ is an *r-shallow minor* of G if there exist pairwise disjoint subsets $V_1, \ldots, V_n$ of $V(G)$ such that each $G[V_i]$ has radius at most r and for all edges $v_i v_j \in E(H)$, $\mathrm{dist}_G(V_i, V_j) = 1$. A class $\mathcal{C}$ of graphs has *bounded expansion* if there is $f : \mathbb{N} \to \mathbb{N}$ such that for all $r \in \mathbb{N}$, $G \in \mathcal{C}$, and an r-shallow minor H of G, $|E(H)|/|V(H)| \leqslant f(r)$. A class $\mathcal{C}$ of graphs is *nowhere dense* if there is $g : \mathbb{N} \to \mathbb{N}$ such that for all $r \in \mathbb{N}$ and $G \in \mathcal{C}$, $K_{g(r)}$ is not an r-shallow minor of G.

For vertices $v \in A$ and $u \in V(G) \backslash A$, a path P from u to v is *A-avoiding* if $V(P) \cap A = \{v\}$. For a vertex $u \in V(G) \backslash A$, the *r-projection* of u on A, denoted by $M_r^G(u, A)$, is the set of all vertices $v \in A$ connected to u by an A-avoiding path of length at most r in G. The *r-projection profile* of u on A is a function $\rho_r^G[u, A] : A \to [r] \cup \{\infty\}$ such that for each vertex $v \in A$, $\rho_r^G[u, A](v)$ is ∞ if there is no A-avoiding path of length at most r from u to v, and otherwise the length of a shortest A-avoiding path from u to v. Let $\mu_r(G, A) := |\{\rho_r^G[u, A] : u \in V(G) \backslash A\}|$. We will use the following lemmas.

▶ **Lemma 2.1** (Eickmeyer et al. [20]). *For every nowhere dense class $\mathcal{C}$ of graphs, there is $f_{\mathrm{proj}} : \mathbb{N} \times \mathbb{R}_+ \to \mathbb{N}$ such that for all $r \in \mathbb{N}$, $\varepsilon > 0$, $G \in \mathcal{C}$, and $X \subseteq V(G)$, $\mu_r(G, X) \leqslant f_{\mathrm{proj}}(r, \varepsilon) \cdot |X|^{1+\varepsilon}$.*

For $t \geqslant 0$, a set $X \subseteq V(G)$ is *(r, t)-close* if $|M_r^G(u, X)| \leqslant t$ for every $u \in V(G) \backslash X$.

▶ **Lemma 2.2** (Eickmeyer et al. [20]). *For every nowhere dense class $\mathcal{C}$ of graphs, there exist $f_{\mathrm{cl}} : \mathbb{N} \times \mathbb{R}_+ \to \mathbb{N}$ and a polynomial-time algorithm that for all $r \in \mathbb{N}$, $\varepsilon > 0$, $G \in \mathcal{C}$, and $X \subseteq V(G)$, outputs an $(r, f_{\mathrm{cl}}(r, \varepsilon) \cdot |X|^{\varepsilon})$-close set $X_{\mathrm{cl}} \supseteq X$ of size at most $f_{\mathrm{cl}}(r, \varepsilon) \cdot |X|^{1+\varepsilon}$.*

For a set $X \subseteq V(G)$, an *r-path closure* of X is a set $X_{\mathrm{pth}} \supseteq X$ such that for $u, v \in X$, if $\mathrm{dist}_G(u, v) \leqslant r$, then $\mathrm{dist}_{G[X_{\mathrm{pth}}]}(u, v) = \mathrm{dist}_G(u, v)$.

▶ **Lemma 2.3** (Eickmeyer et al. [20]). *For every nowhere dense class $\mathcal{C}$ of graphs, there exist $f_{\mathrm{pth}} : \mathbb{N} \times \mathbb{R}_+ \to \mathbb{N}$ and a polynomial-time algorithm that for all $r \in \mathbb{N}$, $\varepsilon > 0$, $G \in \mathcal{C}$, and $X \subseteq V(G)$, outputs an r-path closure of X having size at most $f_{\mathrm{pth}}(r, \varepsilon) \cdot |X|^{1+\varepsilon}$.*

Drange et al. [17] showed analogues of these three lemmas on classes of graphs with bounded expansion. By substituting Lemmas 2.1, 2.2, and 2.3 with their analogues, we can easily obtain linear kernels for $(p, r, \mathcal{F})$-Covering and $(p, r, \mathcal{F})$-Packing on classes of graphs with bounded expansion. Thus, we mainly focus on constructing almost linear kernels for the problems on nowhere dense classes of graphs.

A class $\mathcal{C}$ of graphs is *uniformly quasi-wide* if there exist $N : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and $s : \mathbb{N} \to \mathbb{N}$ such that for all $G \in \mathcal{C}$ and $A \subseteq V(G)$ with $|A| \geqslant N(r, m)$, there exist sets $S \subseteq V(G)$ and $B \subseteq A \backslash S$ such that $|S| \leqslant s(r)$, $|B| \geqslant m$, and B is distance-r independent in $G - S$.

▶ **Theorem 2.4** (Kreutzer, Rabinovich, and Siebertz [33]). *Let $\mathcal{C}$ be a nowhere dense class of graphs. For every $r \geqslant 0$, there are $p(r), s(r)$ such that for all $G \in \mathcal{C}$, $m \in \mathbb{N}$, and $A \subseteq V(G)$ with $|A| \geqslant m^{p(r)}$, there are sets $S \subseteq V(G)$ and $B \subseteq A \backslash S$ such that $|S| \leqslant s(r)$, $|B| \geqslant m$, and B is distance-r independent in $G - S$. Moreover, if K_c is not an r-shallow minor of G, then $s(r) \leqslant c \cdot r$ and one can find desired sets S and B in $O(r \cdot c \cdot |A|^{c+6} \cdot |V(G)|^2)$ time.*

VC-dimension. A *set-system* is a family of subsets of a set, called the *ground set*. Let $\mathcal{S}$ be a set-system with the ground set S. A set $S' \subseteq S$ is *shattered* by $\mathcal{S}$ if $|\{S' \cap T : T \in \mathcal{S}\}| = 2^{|S'|}$. The *Vapnik-Chervonenkis dimension*, or *VC-dimension* for short, of $\mathcal{S}$ is the largest cardinality of a shattered subset of S by $\mathcal{S}$. Observe that if a set $S' \subseteq S$ is shattered by $\mathcal{S}$, then every subset of S' is also shattered by $\mathcal{S}$. In addition, for every $\mathcal{S}' \subseteq \mathcal{S}$, the VC-dimension of $\mathcal{S}'$ is at most that of $\mathcal{S}$.

▶ **Proposition 2.5** (See [34, Proposition 10.3.3]). *Let $F(X_1, \ldots, X_d)$ be a set-theoretic expression using set variables $X_1, \ldots, X_d$ and the operations of union, intersection, and difference. Let $\mathcal{S}$ be a set-system with the ground set S, and $\mathcal{T} := \{F(S_1, \ldots, S_d) : S_1, \ldots, S_d \in \mathcal{S}\}$. If $\mathcal{S}$ has VC-dimension $c < \infty$, then $\mathcal{T}$ has VC-dimension $O(cd \log d)$.*

A *hitting set* of $\mathcal{S}$ is a set $X \subseteq S$ such that for every $T \in \mathcal{S}$, $T \cap X \neq \varnothing$. Let $\tau(\mathcal{S})$ be the minimum size of a hitting set of $\mathcal{S}$.

Brönnimann and Goodrich [8] and Even, Rawitz, and Shahar [21] presented polynomial-time algorithms finding a hitting set X of a nonempty set-system $\mathcal{S}$ having VC-dimension at most c with $|X| = O(c \cdot \tau(\mathcal{S}) \cdot \ln \tau(\mathcal{S}))$.

▶ **Theorem 2.6** ([8, 21]). *There exist a constant C_τ and a polynomial-time algorithm that for every nonempty set-system $\mathcal{S}$ having VC-dimension at most c, outputs a hitting set of $\mathcal{S}$ having size at most $C_\tau \cdot c \cdot \tau(\mathcal{S}) \cdot \ln \tau(\mathcal{S}) + 1$.*

The VC-dimension of G is defined by the VC-dimension of $\{N_G[v] : v \in V(G)\}$.

▶ **Theorem 2.7** (Adler and Adler [2]). *Let $\mathcal{C}$ be a nowhere dense class of graphs and $\phi(x, y)$ be a first-order formula such that for all $G \in \mathcal{C}$ and vertices v and w of G, $G \models \phi(v, w)$ if and only if $G \models \phi(w, v)$. For a graph $G \in \mathcal{C}$, let $G_\phi := (V(G), \{vw : G \models \phi(v, w)\})$. Then there exists a nonnegative integer c depending on $\mathcal{C}$ and ϕ such that every graph in $\{G_\phi : G \in \mathcal{C}\}$ has VC-dimension at most c.*

For every $p \in \mathbb{N}$, the property that the distance between two vertices is at most p can be expressed in a first-order formula, so Theorem 2.7 has the following corollary.

▶ **Corollary 2.8.** *For every nowhere dense class $\mathcal{C}$ of graphs, there exists a function $f_{\mathrm{vc}} : \mathbb{N} \to \mathbb{N}$ such that for all $p \in \mathbb{N}$ and $G \in \mathcal{C}$, G^p has VC-dimension at most $f_{\mathrm{vc}}(p)$.* ◀

3 Kernels for the $(p, r, \mathcal{F})$-Covering problems

Let p, r be nonnegative integers with $p \leqslant 2r + 1$ and let $\mathcal{F}$ be a nonempty finite family of connected graphs. In this section, we present an almost linear kernel for $(p, r, \mathcal{F})$-Covering on every nowhere dense class of graphs. To do this, we divert to an annotated variant of $(p, r, \mathcal{F})$-Covering. For a graph G and a set $A \subseteq V(G)$, a set $D \subseteq V(G)$ is a *$(p, r, \mathcal{F})$-cover* of A in G if there is no set $X \subseteq A \backslash N_G^r[D]$ such that $G^p[X]$ is isomorphic to a graph in $\mathcal{F}$. We denote by $\gamma_{p,r}^{\mathcal{F}}(G, A)$ the minimum size of a $(p, r, \mathcal{F})$-cover of A in G. For a graph G, a set $A \subseteq V(G)$, and an integer k, the Annotated $(p, r, \mathcal{F})$-Covering problem asks whether $\gamma_{p,r}^{\mathcal{F}}(G, A) \leqslant k$.

We first construct an almost linear kernel for Annotated $(p, r, \mathcal{F})$-Covering on every nowhere dense class of graphs. Every instance of $(p, r, \mathcal{F})$-Covering can be seen as an instance of an annotated variant, so we apply the almost linear kernel to the input instance. Afterwards, we construct an equivalent instance of $(p, r, \mathcal{F})$-Covering by attaching a small graph, which will be called a $(p, \mathcal{F})$-critical graph, to the resulting instance obtained from the almost linear kernel.

For a graph G and a set $A \subseteq V(G)$, a *$(p, r, \mathcal{F})$-core* of A in G is a set $Z \subseteq A$ such that every minimum-size $(p, r, \mathcal{F})$-cover of Z in G is a $(p, r, \mathcal{F})$-cover of A in G. Observe that $\gamma^{\mathcal{F}}_{p,r}(G, A) = \gamma^{\mathcal{F}}_{p,r}(G, Z)$ and A is a $(p, r, \mathcal{F})$-core of A in G. We derive an almost linear kernel for Annotated $(p, r, \mathcal{F})$-Covering from Proposition 3.1 saying that we can either confirm that the given instance is a no-instance, or reduce the size of a $(p, r, \mathcal{F})$-core of A in G.

▶ **Proposition 3.1.** *For every nowhere dense class $\mathcal{C}$ of graphs, there is $f_{\mathrm{core}} : \mathbb{N} \times \mathbb{N} \times \mathbb{R}_+ \to \mathbb{N}$ such that for every nonempty family $\mathcal{F}$ of connected graphs with at most d vertices, $p, r \in \mathbb{N}$ with $p \leqslant 2r + 1$, and $\varepsilon > 0$, there is a polynomial-time algorithm that given a graph $G \in \mathcal{C}$, $A \subseteq V(G)$, $k \in \mathbb{N}$, and a $(p, r, \mathcal{F})$-core Z of A in G with $|Z| > f_{\mathrm{core}}(r, d, \varepsilon) \cdot k^{1+\varepsilon}$, either correctly decides that $\gamma^{\mathcal{F}}_{p,r}(G, A) > k$, or outputs a vertex $z \in Z$ such that $Z \setminus \{z\}$ is a $(p, r, \mathcal{F})$-core of A in G.*

We will use the following proposition to prove Proposition 3.1.

▶ **Proposition 3.2.** *For every nowhere dense class $\mathcal{C}$ of graphs, there is $f_{\mathrm{apx}} : \mathbb{N} \times \mathbb{N} \times \mathbb{R}_+ \to \mathbb{N}$ such that for every nonempty family $\mathcal{F}$ of connected graphs with at most d vertices, $p, r \in \mathbb{N}$, and $\varepsilon > 0$, there is a polynomial-time algorithm that given a graph $G \in \mathcal{C}$ and $A \subseteq V(G)$, outputs a $(p, r, \mathcal{F})$-cover of A in G having size at most $f_{\mathrm{apx}}(r, d, \varepsilon) \cdot \gamma^{\mathcal{F}}_{p,r}(G, A)^{1+\varepsilon}$.*

Proof. Let $\mathcal{N} := \{N^r_G[v] : v \in V(G)\}$ and $\mathcal{N}_A := \{N^r_G[v] : v \in A\}$. By Corollary 2.8, $\mathcal{N}$ has VC-dimension at most $f_{\mathrm{vc}}(r)$. Since $\mathcal{N}_A \subseteq \mathcal{N}$, $\mathcal{N}_A$ has VC-dimension at most $f_{\mathrm{vc}}(r)$. Let $\mathcal{H}_0 := \{N^r_G[B] : B \subseteq A,\ |B| \leqslant d\}$. Let $\mathcal{H}_1$ be the family of sets $B \subseteq A$ such that $G^p[B]$ is isomorphic to a graph in $\mathcal{F}$, and $\mathcal{H}_2 := \{N^r_G[B] : B \in \mathcal{H}_1\}$. Since $\mathcal{N}$ has VC-dimension at most $f_{\mathrm{vc}}(r)$, by Proposition 2.5, $\mathcal{H}_0$ has VC-dimension at most $O(f_{\mathrm{vc}}(r) \cdot d \log d)$. Since $\mathcal{H}_2 \subseteq \mathcal{H}_0$, $\mathcal{H}_2$ has VC-dimension at most $O(f_{\mathrm{vc}}(r) \cdot d \log d)$.

Let $\gamma := \gamma^{\mathcal{F}}_{p,r}(G, A)$ and δ be the VC-dimension of $\mathcal{H}_2$. Observe that $(p, r, \mathcal{F})$-covers of A in G correspond to hitting sets of $\mathcal{H}_2$, and vice versa. By Theorem 2.6, one can find in polynomial time a hitting set X of $\mathcal{H}_2$ having size at most $C_\tau \cdot \delta \cdot \gamma \cdot \ln \gamma + 1$. Thus, one can choose the function $f_{\mathrm{apx}}(r, d, \varepsilon)$ with $|X| \leqslant f_{\mathrm{apx}}(r, d, \varepsilon) \cdot \gamma^{1+\varepsilon}$. ◀

Proof of Proposition 3.1. The function $f_{\mathrm{core}}(r, d, \varepsilon)$ will be defined later. At the beginning, we assume that $|Z| > f_{\mathrm{core}}(r, d, \varepsilon) \cdot k^{1+C\varepsilon}$ for some constant C, and at the end, we scale ε accordingly. If Z contains a vertex v such that for every set $B \subseteq Z \setminus \{v\}$ with $|B| \leqslant d - 1$, $G^p[B \cup \{v\}]$ is isomorphic to no graph in $\mathcal{F}$, then the statement holds by taking v as z. Thus, we may assume that for every $v \in Z$, there is a set $B \subseteq Z \setminus \{v\}$ such that $G^p[B \cup \{v\}]$ is isomorphic to a graph in $\mathcal{F}$.

By Proposition 3.2, one can find in polynomial time a $(p, r, \mathcal{F})$-cover X of Z in G having size at most $f_{\mathrm{apx}}(r, d, \varepsilon) \cdot \gamma^{\mathcal{F}}_{p,r}(G, Z)^{1+\varepsilon}$. If $|X| > f_{\mathrm{apx}}(r, d, \varepsilon) \cdot k^{1+\varepsilon}$, then $\gamma^{\mathcal{F}}_{p,r}(G, A) = \gamma^{\mathcal{F}}_{p,r}(G, Z) > k$. Thus, we may assume that $|X| \leqslant f_{\mathrm{apx}}(r, d, \varepsilon) \cdot k^{1+\varepsilon}$. Let $r' := 2pd + 3r$. By Lemma 2.2, one can find in polynomial time an $(r', f_{\mathrm{cl}}(r', \varepsilon) \cdot |X|^{\varepsilon})$-close set $X_{\mathrm{cl}} \supseteq X$ of size at most $f_{\mathrm{cl}}(r', \varepsilon) \cdot |X|^{1+\varepsilon} \leqslant f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{apx}}(r, d, \varepsilon)^{1+\varepsilon} \cdot k^{1+3\varepsilon}$.

Let $\sim$ be an equivalence relation on $Z \setminus X_{\mathrm{cl}}$ such that for vertices $u, v \in Z \setminus X_{\mathrm{cl}}$, $u \sim v$ if and only if $\rho^G_{r'}[u, X_{\mathrm{cl}}] = \rho^G_{r'}[v, X_{\mathrm{cl}}]$. By Lemma 2.1,

$$\mathrm{index}(\sim) \leqslant f_{\mathrm{proj}}(r', \varepsilon) \cdot |X_{\mathrm{cl}}|^{1+\varepsilon} \leqslant f_{\mathrm{proj}}(r', \varepsilon) \cdot f_{\mathrm{cl}}(r', \varepsilon)^{1+\varepsilon} \cdot f_{\mathrm{apx}}(r, d, \varepsilon)^{1+3\varepsilon} \cdot k^{1+7\varepsilon}.$$

Let $p(r')$ and $s := s(r')$ be the constants in Theorem 2.4. Let

$$\xi := 2 \cdot f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{apx}}(r, d, \varepsilon)^{\varepsilon} \cdot k^{2\varepsilon} + d^2/4 + s + 1 \quad \text{and} \quad m := 2^{2^{d^2/2+sd} \cdot (r+1)^{sd}} \cdot \xi + 1.$$

By setting $C = 7 + 2 \cdot p(r')$, one can choose $f_{\mathrm{core}}(r, d, \varepsilon)$ with $f_{\mathrm{core}}(r, d, \varepsilon) \cdot k^{1+C\varepsilon} \geqslant |X_{\mathrm{cl}}| + \mathrm{index}(\sim) \cdot m^{p(r')}$. Since $|Z| > f_{\mathrm{core}}(r, d, \varepsilon) \cdot k^{1+C\varepsilon}$, we have that $|Z \backslash X_{\mathrm{cl}}| > \mathrm{index}(\sim) \cdot m^{p(r')}$. Thus, by the pigeonhole principle, there is an equivalence class λ of $\sim$ with $|\lambda| > m^{p(r')}$. By Theorem 2.4, one can find in polynomial time sets $S \subseteq V(G)$ and $L \subseteq \lambda \backslash S$ such that $|S| \leqslant s$, $|L| \geqslant m$, and L is distance-r' independent in $G - S$.

We are going to find a desired vertex z from L. To do this, we define the following. For each $i \in [d]$, let $\mathcal{G}_i$ be the set of all graphs whose vertex sets are $[i]$. Note that $|\mathcal{G}_i| = 2^{i(i-1)/2}$ for each $i \in [d]$. Let $\mathcal{H}$ be the set of functions $\rho : S \to [2r+1] \cup \{\infty\}$. Since $|S| \leqslant s$, we have that $|\mathcal{H}| \leqslant (2r+2)^s$. For each $i \in [d]$, let $\mathcal{H}_i$ be the set of all vectors $(h_1, \ldots, h_i, g)$ of length $i+1$ where $h_j \in \mathcal{H}$ for each $j \in [i]$ and $g \in \mathcal{G}_i$. Let $\overline{\mathcal{H}} := \bigcup_{i=1}^{d} \mathcal{H}_i$. Note that

$$|\overline{\mathcal{H}}| = \sum_{i=1}^{d} |\mathcal{H}_i| = \sum_{i=1}^{d} (|\mathcal{H}|^i \cdot |\mathcal{G}_i|) \leqslant \sum_{i=1}^{d} ((2r+2)^{si} \cdot 2^{i(i-1)/2}) \leqslant 2^{d^2/2+sd} \cdot (r+1)^{sd}.$$

Let $\ell := |\overline{\mathcal{H}}|$. We take an arbitrary ordering $\sigma_1, \ldots, \sigma_\ell$ of $\overline{\mathcal{H}}$. For each $v \in L$, let $\mathcal{A}_v := \varnothing$ and $\mathbf{x}(v)$ be a zero vector of length ℓ. One can enumerate in polynomial time the sets $B \subseteq Z \backslash \{v\}$ of size at most $d-1$ such that $G^p[B \cup \{v\}]$ is isomorphic to a graph in $\mathcal{F}$. For each such B, we do the following. If there is an index $i \in [\ell]$ such that the i-th entry of $\mathbf{x}(v)$ is 0 and for $\sigma_i = (h_1^i, \ldots, h_t^i, g_i) \in \overline{\mathcal{H}}$, there is an isomorphism $\phi_i : (B \backslash S) \cup \{v\} \to [t]$ between $(G-S)^p[(B \backslash S) \cup \{v\}]$ and g_i where $\phi_i(v) = 1$ and $\rho^G_{2r+1}[\phi_i^{-1}(j), S] = h_j^i$ for each $j \in [t]$, then we put B into $\mathcal{A}_v$ and convert the i-th entry of $\mathbf{x}(v)$ to 1. Otherwise, we do nothing for the chosen B. Since $|B| \leqslant d-1$, one can check in polynomial time whether B satisfies the conditions. Thus, the resulting $\mathcal{A}_v$ and $\mathbf{x}(v)$ can be computed in polynomial time.

For each $v \in L$, since $Z \backslash \{v\}$ has a subset B such that $G^p[B \cup \{v\}]$ is isomorphic to a graph in $\mathcal{F}$, $\mathcal{A}_v \neq \varnothing$ and $\mathbf{x}(v)$ has a nonzero entry. For each set $B \in \mathcal{A}_v$, let B^* be the vertex set of the component of $(G-S)^p[(B \backslash S) \cup \{v\}]$ having v, and $\mathcal{B}_v := \bigcup_{B \in \mathcal{A}_v} B^*$.

Since $|L| \geqslant m = 2^{2^{d^2/2+sd} \cdot (r+1)^{sd}} \cdot \xi + 1$ and $\ell \leqslant 2^{d^2/2+sd} \cdot (r+1)^{sd}$, by the pigeonhole principle, L has a subset κ_1 such that $|\kappa_1| \geqslant \xi + 1$ and $\mathbf{x}(v) = \mathbf{x}(w)$ for all $v, w \in \kappa_1$. Let z be a vertex in κ_1 such that $\mathrm{dist}_{G-S}(\mathcal{B}_z, X_{\mathrm{cl}}) \geqslant \mathrm{dist}_{G-S}(\mathcal{B}_v, X_{\mathrm{cl}})$ for every $v \in \kappa_1$.

We show that $Z \backslash \{z\}$ is a $(p, r, \mathcal{F})$-core of A in G. To do this, for a minimum-size $(p, r, \mathcal{F})$-cover D of $Z \backslash \{z\}$ in G, we need to show that D is a $(p, r, \mathcal{F})$-cover of A in G. Since Z is a $(p, r, \mathcal{F})$-core of A in G, it suffices to show that D is a $(p, r, \mathcal{F})$-cover of Z in G.

Suppose for contradiction that D is not a $(p, r, \mathcal{F})$-cover of Z in G. Since D is a $(p, r, \mathcal{F})$-cover of $Z \backslash \{z\}$ in G, there is a set $B_z \subseteq Z \backslash (N_G^r[D] \cup \{z\})$ such that $G^p[B_z \cup \{z\}]$ is isomorphic to a graph in $\mathcal{F}$. In particular, there exist a graph $H \in \mathcal{G}_t$ for some $t \leqslant d$ and an isomorphism $\psi_z : (B_z \backslash S) \cup \{z\} \to [t]$ between $(G-S)^p[(B_z \backslash S) \cup \{z\}]$ and H where $\psi_z(z) = 1$. For each $v \in \kappa_1 \backslash \{z\}$, there exist $B_v \in \mathcal{A}_v$ and an isomorphism $\psi_v : (B_v \backslash S) \cup \{v\} \to [t]$ between $(G-S)^p[(B_v \backslash S) \cup \{v\}]$ and H where $\psi_v(v) = 1$ and for each $j \in [t]$, $\rho^G_{2r+1}[\psi_v^{-1}(j), S] = \rho^G_{2r+1}[\psi_z^{-1}(j), S]$. To derive a contradiction, we do the following steps.

(1) Find a set $\kappa_3 \subseteq \kappa_1 \backslash \{z\}$ such that for each $u \in \kappa_3$, $\mathrm{dist}_{G-S}(\mathcal{B}_u, X_{\mathrm{cl}}) > r$ and $G^p[B_u^* \cup (B_z \backslash B_z^*)]$ is isomorphic to $G^p[B_z \cup \{z\}]$.

(2) Show that $|D| \geqslant |\kappa_3|$.

(3) Construct a $(p, r, \mathcal{F})$-cover of $Z \backslash \{z\}$ in G having size less than $|D|$.

Since D is a minimum-size $(p, r, \mathcal{F})$-cover of $Z \backslash \{z\}$ in G, these steps derive a contradiction.

Let κ_1' be the set of vertices $v \in \kappa_1$ with $\mathrm{dist}_{G-S}(\mathcal{B}_v, X_{\mathrm{cl}}) \leqslant r$ and let $\kappa_2 := \kappa_1 \backslash \kappa_1'$. We can show that $|\kappa_1'| \leqslant |M_{r'}^G(z, X_{\mathrm{cl}})|$. Thus, $|\kappa_2| \geqslant \xi + 1 - |M_{r'}^G(z, X_{\mathrm{cl}})| \geqslant f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{apx}}(r, d, \varepsilon)^{\varepsilon} \cdot k^{2\varepsilon} + d^2/4 + s + 2$. Since κ_2 is nonempty, by the choice of z, κ_2 contains z. Let B_z^* be the vertex set of the component of $(G-S)^p[(B_z \backslash S) \cup \{z\}]$ having z. Note that for vertices $v, w \in \kappa_2$, $\psi_v^{-1} \circ \psi_w$ is an isomorphism between $(G-S)^p[(B_w \backslash S) \cup \{w\}]$ and $(G-S)^p[(B_v \backslash S) \cup \{v\}]$ assigning w to v. Thus, $\psi_v^{-1} \circ \psi_z(B_z^*) = B_v^*$. The following claim shows that the isomorphism is indeed an isomorphism between induced subgraphs of G^p.

▷ Claim 1. For vertices $v, w \in \kappa_2$, $\psi_w^{-1} \circ \psi_v$ is an isomorphism between $G^p[(B_v \backslash S) \cup \{v\}]$ and $G^p[(B_w \backslash S) \cup \{w\}]$.

Proof. It suffices to show that for $i, j \in [t]$, $\psi_v^{-1}(i)$ is adjacent to $\psi_v^{-1}(j)$ in G^p if and only if $\psi_w^{-1}(i)$ is adjacent to $\psi_w^{-1}(j)$ in G^p. Suppose that $\psi_v^{-1}(i)$ is adjacent to $\psi_v^{-1}(j)$ in G^p. Since $\psi_w^{-1} \circ \psi_v$ is an isomorphism between $(G \backslash S)^p[(B_v \backslash S) \cup \{v\}]$ and $(G \backslash S)^p[(B_w \backslash S) \cup \{w\}]$, we may assume that $\psi_v^{-1}(i)$ and $\psi_v^{-1}(j)$ are nonadjacent in $(G \backslash S)^p[(B_v \backslash S) \cup \{v\}]$. Thus, every path of length at most p in G between $\psi_v^{-1}(i)$ and $\psi_v^{-1}(j)$ has a vertex in S.

We take an arbitrary path Q of G between $\psi_v^{-1}(i)$ and $\psi_v^{-1}(j)$ having length at most p. Let q_i and q_j be the vertices in $V(Q) \cap S$ such that each of $\mathrm{dist}_Q(\psi_v^{-1}(i), q_i)$ and $\mathrm{dist}_Q(\psi_v^{-1}(j), q_j)$ is minimum. Such q_i and q_j exist, because Q has a vertex in S. Let Q_i be the subpath of Q between $\psi_v^{-1}(i)$ and q_i, and Q_j be the subpath of Q between $\psi_v^{-1}(j)$ and q_j. Note that both Q_i and Q_j are S-avoiding paths of length at most $p \leqslant 2r+1$.

Since $\{v, w\} \subseteq \kappa_2 \subseteq \kappa_1$, $\rho_{2r+1}^G[\psi_v^{-1}(i), S]$ and $\rho_{2r+1}^G[\psi_w^{-1}(i), S]$ are same, and therefore G has an S-avoiding path Q_i' between $\psi_w^{-1}(i)$ and q_i whose length is at most that of Q_i. Similarly, G has an S-avoiding path Q_j' between $\psi_w^{-1}(j)$ and q_j whose length is at most that of Q_j. By substituting Q_i and Q_j with Q_i' and Q_j' from Q, respectively, we obtain a walk of G between $\psi_w^{-1}(i)$ and $\psi_w^{-1}(j)$ whose length is at most p. Therefore, $\psi_w^{-1}(i)$ is adjacent to $\psi_w^{-1}(j)$ in G^p. ◁

The following claim shows that except for at most $d^2/4$ vertices in κ_2, for every remaining vertex $u \in \kappa_2$, we can build an isomorphic copy of $G^p[B_z \cup \{z\}]$ by substituting B_z^* with B_u^*.

▷ Claim 2. κ_2 has at most $d^2/4$ vertices v such that $G^p[B_v^* \cup (B_z \backslash B_z^*)]$ is not isomorphic to $G^p[B_z \cup \{z\}]$.

Proof. For vertices $u \in \kappa_2 \backslash \{z\}$ and $i \in \psi_z(B_z^*)$, since $\{u, z\} \subseteq \kappa_2 \subseteq \kappa_1$, $\rho_{2r+1}^G[\psi_u^{-1}(i), S]$ and $\rho_{2r+1}^G[\psi_z^{-1}(i), S]$ are same. Therefore, for each $w \in S$, $\psi_u^{-1}(i)$ is adjacent to w in G^p if and only if $\psi_z^{-1}(i)$ is adjacent to w in G^p. By Claim 1, the restriction of $\psi_u^{-1} \circ \psi_z$ on B_z^* is an isomorphism between $G^p[B_z^*]$ and $G^p[B_u^*]$.

We first show that for all vertices $v \in \kappa_2$, $i \in \psi_z(B_z^*)$, and $w \in B_z \backslash (B_z^* \cup S)$, if $\psi_z^{-1}(i)$ is adjacent to w in G^p, then $\psi_v^{-1}(i)$ is adjacent to w in G^p. Suppose that $\psi_z^{-1}(i)$ is adjacent to w in G^p. Let Q' be a path of G between $\psi_z^{-1}(i)$ and w of length at most p. Since $(G-S)^p[B_z^*]$ is a component of $(G-S)^p[(B_z \backslash S) \cup \{z\}]$ having z and $w \notin B_z^*$, $Q' \cap S \neq \varnothing$.

Let q be the the vertex in $V(Q') \cap S$ such that $\mathrm{dist}_{Q'}(\psi_z^{-1}(i), q)$ is minimum. Such q exists, because Q' has a vertex in S. Let Q_1' be the subpath of Q' between $\psi_z^{-1}(i)$ and q. Note that Q_1' is an S-avoiding path of length at most $p \leqslant 2r+1$. Since $\rho_{2r+1}^G[\psi_v^{-1}(i), S] = \rho_{2r+1}^G[\psi_z^{-1}(i), S]$, there is an S-avoiding path Q_2' in G between $\psi_v^{-1}(i)$ and q having length at most that of Q_1'. By substituting Q_1' with Q_2' from Q', we obtain a walk of G between $\psi_v^{-1}(i)$ and w having length at most p. Thus, $\psi_v^{-1}(i)$ is adjacent to w in G^p. So, there is no pair of $i \in \psi_z(B_z^*)$ and $w \in B_z \backslash (B_z^* \cup S)$ so that in G^p, $\psi_z^{-1}(i)$ is adjacent to w and $\psi_u^{-1}(i)$ is nonadjacent to w.

We now show that if there exist vertices $i \in \psi_z(B_z^*)$ and $w \in B_z \backslash (B_z^* \cup S)$ such that $\psi_z^{-1}(i)$ is nonadjacent to w in G^p, then κ_2 contains at most one vertex x such that $\psi_x^{-1}(i)$ is adjacent to w in G^p. To prove the claim, it suffices to show this statement, because $|B_z^*| \cdot |B_z \backslash (B_z^* \cup S)| \leqslant d^2/4$.

Suppose for contradiction that there exist vertices $i \in \psi_z(B_z^*)$, $w \in B_z \backslash (B_z^* \cup S)$, and distinct $x, x' \in \kappa_2$ such that $\psi_z^{-1}(i)$ is nonadjacent to w in G^p and both $\psi_x^{-1}(i)$ and $\psi_{x'}^{-1}(i)$ are adjacent to w in G^p. Then G has paths R and R' of length at most p from w to $\psi_x^{-1}(i)$ and $\psi_{x'}^{-1}(i)$, respectively. We can verify that R or R' has a vertex in S, as otherwise L is not distance-r' independent in $G - S$. By symmetry, we may assume that R has a vertex in S. Let t be the vertex in $V(R) \cap S$ such that $\mathrm{dist}_R(\psi_x^{-1}(i), t)$ is minimum. Let R_0 be the subpath of R between $\psi_x^{-1}(i)$ and t. Note that R_0 is an S-avoiding path of length at most $p \leqslant 2r + 1$. Since $\rho_{2r+1}^G[\psi_x^{-1}(i), S] = \rho_{2r+1}^G[\psi_z^{-1}(i), S]$, G has an S-avoiding path R_0' between $\psi_z^{-1}(i)$ and t having length at most that of R_0. By substituting R_0 with R_0' from R, we obtain a walk of G between $\psi_z^{-1}(i)$ and w having length at most p, contradicting the assumption that $\psi_z^{-1}(i)$ is nonadjacent to w in G^p, and this proves the claim. ◁

Since $|\kappa_2| \geqslant f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{apx}}(r, d, \varepsilon)^\varepsilon \cdot k^{2\varepsilon} + d^2/4 + s + 2$, by Claim 2, $\kappa_2 \backslash \{z\}$ has a subset κ_3 of size at least $f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{apx}}(r, d, \varepsilon)^\varepsilon \cdot k^{2\varepsilon} + s + 1$ such that for each vertex $u \in \kappa_3$, $G^p[B_u^* \cup (B_z \backslash B_z^*)]$ is isomorphic to $G^p[B_z \cup \{z\}]$, which is isomorphic to a graph in $\mathcal{F}$. This is the end of the first step.

We now show that $|D| \geqslant |\kappa_3|$. For each vertex $u \in \kappa_3$, since $B_u^* \cup (B_z \backslash B_z^*) \subseteq Z \backslash \{z\}$ and D is a $(p, r, \mathcal{F})$-cover of $Z \backslash \{z\}$ in G, there exist vertices $x_u \in B_u^* \cup (B_z \backslash B_z^*)$ and $d_u \in D$ with $\mathrm{dist}_G(x_u, d_u) \leqslant r$. Observe that $x_u \in \psi_u^{-1} \circ \psi_z(B_z^*)$, because $B_z \backslash B_z^* \subseteq B_z \subseteq Z \backslash (N_G^r[D] \cup \{z\})$. Let P_u be an arbitrary path in G between x_u and d_u of length at most r.

▷ Claim 3. For each $u \in \kappa_3$, $V(P_u) \cap (S \cup X_{\mathrm{cl}}) = \varnothing$.

Proof. Let u be a vertex in κ_3. Suppose for contradiction that $V(P_u) \cap S \neq \varnothing$. Let q be the vertex in $V(P_u) \cap S$ such that $\mathrm{dist}_{P_u}(x_u, q)$ is minimum. Let P_1 be the subpath of P_u between x_u and q. Note that P_1 is an S-avoiding path of length at most r. Since $\{u, z\} \subseteq \kappa_2 \subseteq \kappa_1$, G has an S-avoiding path P_2 between $\psi_z^{-1} \circ \psi_u(x_u)$ and q having length at most that of P_1. By substituting P_1 with P_2 from P_u, we obtain a walk of G between $\psi_z^{-1} \circ \psi_u(x_u) \in B_z^* \subseteq B_z \cup \{z\}$ and d_u having length at most r, contradicting the assumption that $B_z \cap N_G^r[D] = \varnothing$. Hence, $V(P_u) \cap S = \varnothing$.

Since $u \notin \kappa_1'$ and $B_u \in \mathcal{A}_u$, we have that

$$\mathrm{dist}_{G \backslash S}(x_u, X_{\mathrm{cl}}) \geqslant \mathrm{dist}_{G \backslash S}(B_u^*, X_{\mathrm{cl}}) \geqslant \mathrm{dist}_{G \backslash S}(\mathcal{B}_u, X_{\mathrm{cl}}) > r.$$

Since P_u is a path of $G \backslash S$ having length at most r, $V(P_u) \cap X_{\mathrm{cl}} = \varnothing$. ◁

We now derive $|D| \geqslant |\kappa_3|$ from the following.

▷ Claim 4. For distinct $u, u' \in \kappa_3$, the vertices d_u and $d_{u'}$ are distinct.

As the last step, we now construct a $(p, r, \mathcal{F})$-cover of $Z \backslash \{z\}$ in G having size less than $|D|$. Let $D_{\mathrm{sell}} := \{d_u : u \in \kappa_3\}$, $D_{\mathrm{buy}} := M_{r'}^G(z, X_{\mathrm{cl}}) \cup S$, and $D' := (D \backslash D_{\mathrm{sell}}) \cup D_{\mathrm{buy}}$. By Claim 4,

$$|D_{\mathrm{sell}}| = |\kappa_3| \geqslant f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{apx}}(r, d, \varepsilon)^\varepsilon \cdot k^{2\varepsilon} + s + 1,$$
$$|D_{\mathrm{buy}}| \leqslant |M_{r'}^G(z, X_{\mathrm{cl}})| + |S| \leqslant f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{apx}}(r, d, \varepsilon)^\varepsilon \cdot k^{2\varepsilon} + s.$$

Since $D_{\mathrm{sell}} \subseteq D$, we have that $|D'| < |D|$. For a contradiction, we show the following claim.

▷ Claim 5. D' is a $(p, r, \mathcal{F})$-cover of $Z \backslash \{z\}$ in G.

Proof. Suppose not. Then there is a set $B' \subseteq Z \backslash (N_G^r[D'] \cup \{z\})$ such that $G^p[B']$ is isomorphic to a graph in $\mathcal{F}$. Since D is a $(p, r, \mathcal{F})$-cover of $Z \backslash \{z\}$ in G and $D \backslash D' \subseteq D_{\text{sell}}$, D_{sell} contains a vertex d_u for some $u \in \kappa_3$ with $\text{dist}_G(d_u, B') \leqslant r$.

Since $(G \backslash S)^p[B_u^*]$ is connected and $|B_u^*| \leqslant d$, $G \backslash S$ has a path Q_0 of length at most $p(d-1)$ between u and x_u. More specifically, Q_0 is a concatenation of paths $Q_0^1, \ldots, Q_0^{t_1}$ for $t_1 \leqslant d-1$ such that for each $i \in [t_1]$, the length of Q_0^i is at most p and the ends of Q_0^i are in B_u^*. Since $\text{dist}_G(d_u, B') \leqslant r$, G has a path Q_1 of length at most r between d_u and $w_1 \in B'$. Since X_{cl} is a $(p, r, \mathcal{F})$-cover of Z in G, G has a path Q_2 of length at most r between $w_2 \in B'$ and $x \in X_{\text{cl}}$. Since $G^p[B']$ is isomorphic to a connected graph in $\mathcal{F}$ and $|B'| \leqslant d$, G has a path R of length at most $p(d-1)$ between w_1 and w_2. More specifically, R is a concatenation of paths $R_1, \ldots, R_{t_2}$ for $t_2 \leqslant d-1$ such that for each $i \in [t_2]$, the length of R_i is at most p and the ends of R_i are in B'. By concatenating Q_0, P_u, Q_1, R, and Q_2, we obtain a walk of G between u and x having length at most

$$\begin{aligned}&|E(Q_0)| + |E(P_u)| + |E(Q_1)| + |E(R)| + |E(Q_2)| \\ &\leqslant p(d-1) + r + r + p(d-1) + r = 2p(d-1) + 3r \leqslant r'.\end{aligned}$$

Let P be a path of G between u and x consisting of edges of the walk. Let b be the vertex in $V(P) \cap (S \cup X_{\text{cl}})$ such that $\text{dist}_P(u, b)$ is minimum. Such b exists, because $x \in X_{\text{cl}}$.

We first show that $\text{dist}_G(b, B') \leqslant r$. Note that Q_0 has no vertex in S. Since $u \notin \kappa_1'$, $\text{dist}_{G \backslash S}(\mathcal{B}_u, X_{\text{cl}}) > r$. Since $p \leqslant 2r+1$, for some $j \in [t_1]$, if Q_0^j has a vertex in X_{cl}, then $\text{dist}_{G \backslash S}(\mathcal{B}_u, X_{\text{cl}}) \leqslant \text{dist}_{G \backslash S}(B_u^*, X_{\text{cl}}) \leqslant r$, a contradiction. Therefore, Q_0 has no vertex in X_{cl}. By Claim 3, $V(P_u) \cap (S \cup X_{\text{cl}}) = \varnothing$. These imply that $b \in V(Q_1) \cup V(R) \cup V(Q_2)$. If $b \in V(Q_1) \cup V(Q_2)$, then $\text{dist}_G(b, B') \leqslant r$ clearly. Since $p \leqslant 2r+1$, for some $j \in [t_2]$, if $b \in R_j$, then $\text{dist}_G(b, B') \leqslant r$. Therefore, $\text{dist}_G(b, B') \leqslant r$.

Since $B' \subseteq Z \backslash (N_G^r[D'] \cup \{z\})$, b is not contained in D'. Since $S \subseteq D_{\text{buy}} \subseteq D'$, b is contained in $X_{\text{cl}} \backslash S$. Since the subpath of P between u and b is an X_{cl}-avoiding path of length at most r', b is contained in $M_{r'}^G(u, X_{\text{cl}})$. Since $\{u, z\} \subseteq \kappa_2 \subseteq \lambda$ where λ is an equivalence class of $\sim$, $M_{r'}^G(u, X_{\text{cl}})$ and $M_{r'}^G(z, X_{\text{cl}})$ are same. Therefore, $b \in M_{r'}^G(z, X_{\text{cl}}) \subseteq D'$, a contradiction. ◁

Claim 5 contradicts the assumption that D is a minimum-size $(p, r, \mathcal{F})$-cover of $Z \backslash \{z\}$ in G. Thus, $Z \backslash \{z\}$ is a $(p, r, \mathcal{F})$-core of A in G. We conclude the proof by scaling ε to ε / C. ◀

After recursively applying Proposition 3.1, started from A, we may assume that we are given a small $(p, r, \mathcal{F})$-core Z. By taking a $(2r+1)$-path closure Y of some superset of Z with Lemma 2.3, we can derive an almost linear kernel for Annotated $(p, r, \mathcal{F})$-Covering as follows.

▶ **Theorem 3.3.** *For every nowhere dense class $\mathcal{C}$ of graphs, there is $f_{\text{cov}} : \mathbb{N} \times \mathbb{N} \times \mathbb{R}_+ \to \mathbb{N}$ such that for every nonempty family $\mathcal{F}$ of connected graphs with at most d vertices, $p, r \in \mathbb{N}$ with $p \leqslant 2r+1$, and $\varepsilon > 0$, there is a polynomial-time algorithm that given a graph $G \in \mathcal{C}$, $A \subseteq V(G)$, and $k \in \mathbb{N}$, either correctly decides that $\gamma_{p,r}^{\mathcal{F}}(G, A) > k$, or outputs sets $Y \subseteq V(G)$ of size at most $f_{\text{cov}}(r, d, \varepsilon) \cdot k^{1+\varepsilon}$ and $Z \subseteq A \cap Y$ such that $\gamma_{p,r}^{\mathcal{F}}(G[Y], Z) = \gamma_{p,r}^{\mathcal{F}}(G, A)$.*

We now convert the resulting instance of Theorem 3.3 to an equivalent instance of $(p, r, \mathcal{F})$-Covering. To do this, we will use the following definition and lemmas. For an integer $q \geqslant 0$ and a nonempty family $\mathcal{G}$ of graphs, a graph H is $(q, \mathcal{G})$-*critical* if either

- H is a 1-vertex graph and $\mathcal{G}$ contains a 1-vertex graph, or
- H has at least two vertices, H^q has an induced subgraph isomorphic to a graph in $\mathcal{G}$, and for every vertex v of H, $(H-v)^q$ has no induced subgraph isomorphic to a graph in $\mathcal{G}$.

▶ **Lemma 3.4.** *Let $\mathcal{G}$ be a nonempty family of graphs. Let F be a graph in $\mathcal{G}$ and d be the order of F. For every positive integer q, there is a $(q,\mathcal{G})$-critical graph of order at most $d(dq+1)/2$. Moreover, if every graph in $\mathcal{G}$ has order at most d, then one can construct the $(q,\mathcal{G})$-critical graph in time polynomial in d.*

Proof. Let F_0 be the q-subdivision of F. Since F has at most $d(d-1)/2$ edges,

$$|V(F_0)| \leqslant d + \frac{d(d-1)(q-1)}{2} = d \cdot \frac{dq-d-q+3}{2} \leqslant \frac{d(dq+1)}{2}.$$

Let H be a graph which is initially set as F_0. Note that H^q has an induced subgraph isomorphic to $F \in \mathcal{G}$. If $|V(H)| = 1$, then H is $(q,\mathcal{G})$-critical. Otherwise, for each vertex v of H, we check whether $(H\backslash v)^q$ has an induced subgraph isomorphic to a graph in $\mathcal{G}$. If H has no such vertex, then H is $(q,\mathcal{G})$-critical. Otherwise, we set H by $H\backslash v$ and do the above process until either $|V(H)| = 1$ or H has no such a vertex. It is readily seen that the resulting graph is $(q,\mathcal{G})$-critical graph and has at most $d(dq+1)/2$ vertices. Whole these processes work in polynomial time when every graph in $\mathcal{G}$ has at most d vertices. ◀

The following lemma shows that every vertex of a $(q,\mathcal{G})$-critical graph is a $(q,\lfloor q/2\rfloor,\mathcal{G})$-cover of it.

▶ **Lemma 3.5.** *Let $\mathcal{G}$ be a nonempty family of graphs, and q be a positive integer. If H is a $(q,\mathcal{G})$-critical graph and there is a set $B \subseteq V(H)$ such that $H^q[B]$ is isomorphic to a graph in $\mathcal{G}$, then for every $x \in V(H)$, B contains a vertex in $N_H^{\lfloor q/2\rfloor}[x]$.*

Proof. Suppose for contradiction that B contains no vertex in $N_H^{\lfloor q/2\rfloor}[x]$. Since H is $(q,\mathcal{G})$-critical, $(H\backslash x)^q[B]$ is isomorphic to no graph in $\mathcal{G}$. Since $H^q[B]$ is isomorphic to a graph in $\mathcal{G}$, B contains distinct vertices v and w such that v and w are adjacent in H^q and every path of H between v and w having length at most q should contain x. However, since neither v nor w is in $N_H^{\lfloor q/2\rfloor}[x]$, if H has a path P between v and w having x as an internal vertex, then the length of P is at least $2\lfloor q/2\rfloor + 2 > q$, a contradiction. ◀

To prove Theorem 1.1, we construct an equivalent instance of $(p,r,\mathcal{F})$-Covering by attaching a $(p,\mathcal{F})$-critical graph to the resulting instance of Theorem 3.3.

Sketch of the proof of Theorem 1.1. The cases where either $r=0$ or $p=0$ are relatively easy to deal with. Thus, in this sketch, we assume that both r and p are positive. Let d be the maximum order of a graph in $\mathcal{F}$. By Lemma 3.4, one can find in polynomial time a $(p,\mathcal{F})$-critical graph H having at most $d(dp+1)/2$ vertices. Let $p' := \lfloor p/2\rfloor$ and x be a vertex of H. We construct the graph G' as follows: take the disjoint union of $G[Y]$ and H, add a new vertex h, and for each vertex $v \in (Y\backslash Z) \cup N_H^{p'}[x]$, connect h and v by a path P_v of length r. We can show that the resulting graph G' is the desired one by Lemma 3.5. ◀

4 Kernels for the $(p,r,\mathcal{F})$-Packing problems

Let p,r be nonnegative integers with $p \leqslant r+1$ and $\mathcal{F}$ be a nonempty finite family of connected graphs. We present an almost linear kernel for $(p,r,\mathcal{F})$-Packing on every nowhere dense class of graphs. We also divert to the annotated variant of $(p,r,\mathcal{F})$-Packing. Although

the proof scheme is similar to that of the kernel for $(p,r,\mathcal{F})$-Covering, we need a more intricate approximation algorithm, and the key lemma and the main steps of its proof are quite different from those of $(p,r,\mathcal{F})$-Covering.

For a graph G and a set $A \subseteq V(G)$, a $(p,r,\mathcal{F})$-*packing* of A in G is a family of subsets of A, say $A_1,\dots,A_\ell$ such that each $G^p[A_i]$ is isomorphic to a graph in $\mathcal{F}$, and for all $1 \leqslant i < j \leqslant \ell$, $\mathrm{dist}_G(A_i,A_j) > r$. We denote by $\alpha^{\mathcal{F}}_{p,r}(G,A)$ the maximum size of a $(p,r,\mathcal{F})$-packing of A in G. For a graph G, a set $A \subseteq V(G)$, and an integer k, Annotated $(p,r,\mathcal{F})$-Packing asks whether $\alpha^{\mathcal{F}}_{p,r}(G,A) \geqslant k$. We first derive an almost linear kernel for this problem.

▶ **Proposition 4.1.** *For every nowhere dense class $\mathcal{C}$ of graphs, there is $f_{\mathrm{rd}} : \mathbb{N}\times\mathbb{N}\times\mathbb{R}_+ \to \mathbb{N}$ such that for every nonempty family $\mathcal{F}$ of connected graphs with at most d vertices, $p,r \in \mathbb{N}$ with $p \leqslant 2\lfloor r/2\rfloor+1$, and $\varepsilon > 0$, there is a polynomial-time algorithm that given a graph $G \in \mathcal{C}$, $A \subseteq V(G)$ with $|A| > f_{\mathrm{rd}}(r,d,\varepsilon)\cdot k^{1+\varepsilon}$, and $k \in \mathbb{N}$, either correctly decides that $\alpha^{\mathcal{F}}_{p,r}(G,A) > k$, or outputs a vertex $z \in A$ such that $\alpha^{\mathcal{F}}_{p,r}(G,A) \geqslant k$ if and only if $\alpha^{\mathcal{F}}_{p,r}(G,A\backslash\{z\}) \geqslant k$.*

▶ **Proposition 4.2.** *For every nowhere dense class $\mathcal{C}$ of graphs, there is $f_{\mathrm{dual}} : \mathbb{N}\times\mathbb{N}\times\mathbb{R}_+ \to \mathbb{N}$ such that for every nonempty family $\mathcal{F}$ of connected graphs with at most d vertices, $p,r,r_0 \in \mathbb{N}$ with $\max\{p,r_0\} \leqslant 2r+1$, and $\varepsilon > 0$, there is a polynomial-time algorithm that given a graph $G \in \mathcal{C}$ and $A \subseteq V(G)$, outputs a $(p,r,\mathcal{F})$-cover of A in G having size at most $f_{\mathrm{dual}}(r,d,\varepsilon)\cdot\alpha^{\mathcal{F}}_{p,r_0}(G,A)^{1+\varepsilon}$.*

Proof of Proposition 4.1. The function $f_{\mathrm{rd}}(r,d,\varepsilon)$ will be defined later. At the beginning, we assume that $|A| > f_{\mathrm{rd}}(r,d,\varepsilon)\cdot k^{1+C\varepsilon}$ for some constant C, and at the end, we scale ε accordingly. We may assume that for every $v \in A$, there is $B \subseteq A\backslash\{v\}$ such that $G^p[B\cup\{v\}]$ is isomorphic to a graph in $\mathcal{F}$. Since $p \leqslant 2\lfloor r/2\rfloor+1$, by Proposition 4.2, one can find a $(p,\lfloor r/2\rfloor,\mathcal{F})$-cover X of A in G having size at most $f_{\mathrm{dual}}(\lfloor r/2\rfloor,d,\varepsilon)\cdot\alpha^{\mathcal{F}}_{p,r}(G,A)^{1+\varepsilon}$ in polynomial time. If $|X| > f_{\mathrm{dual}}(\lfloor r/2\rfloor,d,\varepsilon)\cdot k^{1+\varepsilon}$, then $\alpha^{\mathcal{F}}_{p,r}(G,A) > k$. Thus, we may assume that $|X| \leqslant f_{\mathrm{dual}}(\lfloor r/2\rfloor,d,\varepsilon)\cdot k^{1+\varepsilon}$. Let $r' := 4pd+3r$. By Lemma 2.2, one can find an $(r', f_{\mathrm{cl}}(r',\varepsilon)\cdot|X|^{\varepsilon})$-close set $X_{\mathrm{cl}} \supseteq X$ of size at most $f_{\mathrm{cl}}(r',\varepsilon)\cdot|X|^{1+\varepsilon} \leqslant f_{\mathrm{cl}}(r',\varepsilon)\cdot f_{\mathrm{dual}}(\lfloor r/2\rfloor,d,\varepsilon)^{1+\varepsilon}\cdot k^{1+3\varepsilon}$ in polynomial time.

We define an equivalence relation $\sim$ on $A\backslash X_{\mathrm{cl}}$ such that for $u,v \in A\backslash X_{\mathrm{cl}}$, $u \sim v$ if and only if $\rho^G_{r'}[u,X_{\mathrm{cl}}] = \rho^G_{r'}[v,X_{\mathrm{cl}}]$. Then $\mathrm{index}(\sim) \leqslant f_{\mathrm{proj}}(r',\varepsilon)f_{\mathrm{cl}}(r',\varepsilon)^{1+\varepsilon}f_{\mathrm{dual}}(\lfloor r/2\rfloor,d,\varepsilon)^{1+3\varepsilon}k^{1+7\varepsilon}$ by Lemma 2.1. Let $p(r')$ and $s := s(r')$ be the constants in Theorem 2.4. Let

$$\xi := d\cdot(f_{\mathrm{cl}}(r',\varepsilon)\cdot f_{\mathrm{dual}}(\lfloor r/2\rfloor,d,\varepsilon)^{\varepsilon}\cdot k^{2\varepsilon}+s+d^2/4+1) \quad\text{and}\quad m := 2^{2^{d^2/2}\cdot(r+2)^{sd}}\cdot\xi+1.$$

By setting $C = 7+2\cdot p(r')$, one can choose $f_{\mathrm{rd}}(r,d,\varepsilon)$ with $f_{\mathrm{rd}}(r,d,\varepsilon)\cdot k^{1+C\varepsilon} \geqslant |X_{\mathrm{cl}}| + \mathrm{index}(\sim)\cdot m^{p(r')}$. Since $|A| > f_{\mathrm{rd}}(r,d,\varepsilon)\cdot k^{1+C\varepsilon}$, we have that $|A\backslash X_{\mathrm{cl}}| > \mathrm{index}(\sim)\cdot m^{p(r')}$. Thus, by the pigeonhole principle, there is an equivalence class λ of $\sim$ with $|\lambda| > m^{p(r')}$. By Theorem 2.4, one can find in polynomial time sets $S \subseteq V(G)$ and $L \subseteq \lambda\backslash S$ such that $|S| \leqslant s$, $|L| \geqslant m$, and L is distance-r' independent in $G-S$.

We are going to find a desired vertex z from L. To do this, we define the following. For each $i \in [d]$, let $\mathcal{G}_i$ be the set of all graphs whose vertex sets are $[i]$. Note that $|\mathcal{G}_i| = 2^{i(i-1)/2}$ for each $i \in [d]$. Let $\mathcal{H}'$ be the set of functions $\rho : S \to [r+1]\cup\{\infty\}$. Since $|S| \leqslant s$, we have that $|\mathcal{H}'| \leqslant (r+2)^s$. For each $i \in [d]$, let $\mathcal{H}'_i$ be the set of all vectors $(h_1,\dots,h_i,g)$ of length $i+1$ where $h_j \in \mathcal{H}'$ for each $j \in [i]$ and $g \in \mathcal{G}_i$. Let $\overline{\mathcal{H}'} := \bigcup_{i=1}^{d}\mathcal{H}'_i$. Similar to the proof of Proposition 3.1, we can show that $|\overline{\mathcal{H}'}| \leqslant 2^{d^2/2}\cdot(r+2)^{sd}$. Let $\ell := |\overline{\mathcal{H}'}|$. We take an arbitrary ordering $\sigma_1,\dots,\sigma_\ell$ of $\overline{\mathcal{H}'}$. For each $v \in L$, let $\mathcal{A}_v := \varnothing$ and $\mathbf{x}(v)$ be a zero vector of length ℓ. One can enumerate in polynomial time the sets $B \subseteq A\backslash\{v\}$ of size at most $d-1$ such that $G^p[B\cup\{v\}]$ is isomorphic to a graph in $\mathcal{F}$ in polynomial time. For each

such B, we do the following. If there is an index $i \in [\ell]$ such that the i-th entry of $\mathbf{x}(v)$ is 0 and for $\sigma_i = (h_1^i, \ldots, h_t^i, g_i) \in \overline{\mathcal{H}'}$, there is an isomorphism $\phi_i : (B \backslash S) \cup \{v\} \to [t]$ between $(G-S)^p[(B\backslash S) \cup \{v\}]$ and g_i where $\phi_i(v) = 1$ and for each $j \in [t]$, $\rho_r^G[\phi^{-1}(j), S] = h_j^i$, then we put B into $\mathcal{A}_v$ and convert the i-th entry of $\mathbf{x}(v)$ to 1. Otherwise, we do nothing for the chosen B. Since $|B| \leqslant d-1$, one can check in polynomial time whether B satisfies the conditions. Thus, the resulting $\mathcal{A}_v$ and $\mathbf{x}(v)$ can be computed in polynomial time.

For each $v \in L$, since $A\backslash\{v\}$ has a subset B such that $G^p[B \cup \{v\}]$ is isomorphic to a graph in $\mathcal{F}$, $\mathcal{A}_v \neq \varnothing$ and $\mathbf{x}(v)$ has a nonzero entry. For each $B \in \mathcal{A}_v$, let B^* be the vertex set of the component of $(G-S)^p[(B\backslash S) \cup \{v\}]$ having v. Since $|L| \geqslant m = 2^{2^{d^2/2} \cdot (r+2)^{sd}} \cdot \xi + 1$ and $\ell \leqslant 2^{d^2/2} \cdot (r+2)^{sd}$, by the pigeonhole principle, L has a subset κ_1 such that $|\kappa_1| \geqslant \xi + 1$ and $\mathbf{x}(v) = \mathbf{x}(w)$ for all $v, w \in \kappa_1$. Let z be an arbitrary vertex in κ_1.

We show that $\alpha_{p,r}^{\mathcal{F}}(G, A) \geqslant k$ if and only if $\alpha_{p,r}^{\mathcal{F}}(G, A\backslash\{z\}) \geqslant k$. The backward direction is obvious. Suppose that G has a $(p, r, \mathcal{F})$-packing I of A in G having size at least k. We may assume that z is contained in some $B_z \in I$, because otherwise I is also a $(p, r, \mathcal{F})$-packing of $A\backslash\{z\}$. In particular, there exist a graph $H \in \mathcal{G}_t$ for some $t \leqslant d$ and an isomorphism $\psi_z : (B_z\backslash S) \cup \{z\} \to [t]$ between $(G-S)^p[(B_z\backslash S) \cup \{z\}]$ and H where $\psi_z(z) = 1$. To show that $\alpha_{p,r}^{\mathcal{F}}(G, A\backslash\{z\}) \geqslant k$, it suffices to show that there exist a vertex $z' \in \kappa_1\backslash\{z\}$ and a set $B_{z'} \subseteq A\backslash\{z\}$ such that $z' \in B_{z'}$ and $(I\backslash\{B_z\}) \cup \{B_{z'}\}$ is a $(p, r, \mathcal{F})$-packing of $A\backslash\{z\}$ in G having the same size as I.

Suppose for contradiction that no such z' exists. It means that for each $v \in \kappa_1\backslash\{z\}$, if $A\backslash\{z\}$ has a subset B such that $v \in B$ and $G^p[B]$ is isomorphic to a graph in $\mathcal{F}$, then $I\backslash\{B_z\}$ contains an element B' with $\mathrm{dist}_G(B, B') \leqslant r$, because otherwise we can substitute B_z with B from I. For each $v \in \kappa_1\backslash\{z\}$, there exist $B_v \in \mathcal{A}_v$ and an isomorphism $\psi_v : (B_v\backslash S) \cup \{v\} \to [t]$ between $(G-S)^p[(B_v\backslash S) \cup \{v\}]$ and H where $\psi_v(v) = 1$ and for each $j \in [t]$, $\rho_{r+1}^G[\psi_v^{-1}(j), S] = \rho_{r+1}^G[\psi_z^{-1}(j), S]$. For each $v \in \kappa_1\backslash\{z\}$, let $f(v) := B_v^* \cup (B_z\backslash B_z^*)$.

To derive a contradiction, we do the following steps.

(1) Find a set $\kappa_4 \subseteq \kappa_1\backslash\{z\}$ such that for each $u \in \kappa_4$, $G^p[f(u)]$ is isomorphic to $G^p[B_z \cup \{z\}]$ and I contains an element C_u with $\mathrm{dist}_G(f(u), C_u) \leqslant r$ and $\mathrm{dist}_G(C_u, S) > \lfloor r/2 \rfloor$.

(2) Show that κ_4 contains distinct vertices v and v' with $\mathrm{dist}_{G-S}(v, v') \leqslant r'$.

Since $\kappa_4 \subseteq L$ is distance-r' independent in $G-S$, these steps derive a contradiction.

Let B_z^* be the vertex set of the component of $(G-S)^p[(B_z\backslash S) \cup \{z\}]$ having z. Note that for vertices $v, w \in \kappa_1$, $\psi_v^{-1} \circ \psi_w$ is an isomorphism between $(G-S)^p[(B_w\backslash S) \cup \{w\}]$ and $(G-S)^p[(B_v\backslash S) \cup \{v\}]$ assigning w to v. Thus, $\psi_v^{-1} \circ \psi_z(B_z^*) = B_v^*$.

For the first step, we will use the following three claims. The proofs of Claims 6 and 7 are similar to those of Claims 1 and 2, respectively.

▷ Claim 6. For vertices $v, w \in \kappa_1$, $\psi_w^{-1} \circ \psi_v$ is an isomorphism between $G^p[(B_v\backslash S) \cup \{v\}]$ and $G^p[(B_w\backslash S) \cup \{w\}]$.

▷ Claim 7. κ_1 has at most $d^2/4$ vertices v where $G^p[f(v)]$ is not isomorphic to $G^p[B_z]$.

Proof. For vertices $u \in \kappa_1\backslash\{z\}$ and $i \in \psi_z(B_z^*)$, since $\{u, z\} \subseteq \kappa_1$, $\rho_{r+1}^G[\psi_u^{-1}(i), S]$ and $\rho_{r+1}^G[\psi_z^{-1}(i), S]$ are same. Therefore, for each $w \in S$, $\psi_u^{-1}(i)$ is adjacent to w in G^p if and only if $\psi_z^{-1}(i)$ is adjacent to w in G^p. By Claim 6, the restriction of $\psi_u^{-1} \circ \psi_z$ on B_z^* is an isomorphism between $G^p[B_z^*]$ and $G^p[B_u^*]$.

We first show that for all vertices $v \in \kappa_1$, $i \in \psi_z(B_z^*)$, and $w \in B_z\backslash(B_z^* \cup S)$, if $\psi_z^{-1}(i)$ is adjacent to w in G^p, then $\psi_v^{-1}(i)$ is adjacent to w in G^p. Suppose that $\psi_z^{-1}(i)$ is adjacent to w in G^p. We take an arbitrary path Q' of G between $\psi_z^{-1}(i)$ and w having length at most p. Since $(G\backslash S)^p[B_z^*]$ is a component of $(G\backslash S)^p[(B_z\backslash S) \cup \{z\}]$ having z and $w \notin B_z^*$, Q' must have a vertex in S.

Let q be the the vertex in $V(Q') \cap S$ such that $\mathrm{dist}_{Q'}(\psi_z^{-1}(i), q)$ is minimum. Let Q'_1 be the subpath of Q' between $\psi_z^{-1}(i)$ and q. Note that Q'_1 is an S-avoiding path of length at most $p \leqslant r+1$. Since $\rho^G_{r+1}[\psi_v^{-1}(i), S] = \rho^G_{r+1}[\psi_z^{-1}(i), S]$, G has an S-avoiding path Q'_2 between $\psi_v^{-1}(i)$ and q having length at most that of Q'_1. By substituting Q'_1 with Q'_2 from Q', we obtain a walk of G between $\psi_v^{-1}(i)$ and w having length at most p. Hence, $\psi_v^{-1}(i)$ is adjacent to w in G^p.

Thus, there is no pair of vertices $i \in \psi_z(B^*_z)$ and $w \in B_z \backslash (B^*_z \cup S)$ such that $\psi_z^{-1}(i)$ is adjacent to w in G^p and $\psi_u^{-1}(i)$ is nonadjacent to w in G^p.

We now show that if there exist vertices $i \in \psi_z(B^*_z)$ and $w \in B_z \backslash (B^*_z \cup S)$ such that $\psi_z^{-1}(i)$ is nonadjacent to w in G^p, then κ_1 contains at most one vertex x such that $\psi_x^{-1}(i)$ is adjacent to w in G^p. To prove the claim, it suffices to show this statement, because $|B^*_z| \cdot |B_z \backslash (B^*_z \cup S)| \leqslant d^2/4$.

Suppose for contradiction that there exist $i \in \psi_z(B^*_z)$, $w \in B_z \backslash (B^*_z \cup S)$, and distinct $x, x' \in \kappa_1$ such that $\psi_z^{-1}(i)$ is nonadjacent to w in G^p and both $\psi_x^{-1}(i)$ and $\psi_{x'}^{-1}(i)$ are adjacent to w in G^p. Then G has paths R and R' of length at most p from w to $\psi_x^{-1}(i)$ and $\psi_{x'}^{-1}(i)$, respectively.

We first verify that R or R' has a vertex in S. Suppose not. Since $|B^*_x| \leqslant d$, $G \backslash S$ has a path R_1 of length at most $p(d-1)$ between x and $\psi_x^{-1}(i)$. Similarly, $G \backslash S$ has a path R'_1 of length at most $p(d-1)$ between x' and $\psi_{x'}^{-1}(i)$. Since neither R nor R' has a vertex in S, by concatenating R_1, R, R', and R'_1, we obtain a walk of $G \backslash S$ of length at most $2pd \leqslant r'$ between x and x', contradicting the assumption that L is distance-r' independent in $G \backslash S$. Hence, R or R' has a vertex in S. By symmetry, we may assume that R has a vertex in S.

Let t be the vertex in $V(R) \cap S$ such that $\mathrm{dist}_R(\psi_x^{-1}(i), t)$ is minimum. Let R_0 be the subpath of R between $\psi_x^{-1}(i)$ and t. Note that R_0 is an S-avoiding path of length at most $p \leqslant r+1$. Since $\rho^G_{r+1}[\psi_x^{-1}(i), S] = \rho^G_{r+1}[\psi_z^{-1}(i), S]$, G has an S-avoiding path R'_0 between $\psi_z^{-1}(i)$ and t having length at most that of R_0. By substituting R_0 with R'_0 from R, we obtain a walk of G between $\psi_z^{-1}(i)$ and w having length at most p, contradicting the assumption that $\psi_z^{-1}(i)$ is nonadjacent to w in G^p, and this proves the claim. ◁

Since $|\kappa_1| \geqslant d \cdot (f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{dual}}(\lfloor r/2 \rfloor, d, \varepsilon)^{\varepsilon} \cdot k^{2\varepsilon} + s + d^2/4 + 1) + 1$, by Claim 7, $\kappa_1 \backslash \{z\}$ has a subset κ_2 of size at least $d \cdot (f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{dual}}(\lfloor r/2 \rfloor, d, \varepsilon)^{\varepsilon} \cdot k^{2\varepsilon} + s + 1)$ such that for each vertex $u \in \kappa_2$, $G^p[f(u)]$ is isomorphic to $G^p[B_z]$, which is isomorphic to a graph in $\mathcal{F}$.

For each $u \in \kappa_2$, since $f(u) \subseteq A \backslash \{z\}$, by assumption, $I \backslash \{B_z\}$ contains an element C_u with $\mathrm{dist}_G(f(u), C_u) \leqslant r$. We take an arbitrary path P_u of G between $b_u \in f(u)$ and $c_u \in C_u$ having length at most r. Since $\{B_z, C_u\} \subseteq I$ which is a $(p, r, \mathcal{F})$-packing of A in G, $\mathrm{dist}_G(B_z \backslash B^*_z, C_u) \geqslant \mathrm{dist}_G(B_z, C_u) > r$. Thus, $b_u \in f(u) \backslash (B_z \backslash B^*_z) = B^*_u$.

▷ Claim 8. For each $u \in \kappa_2$, $V(P_u) \cap S = \varnothing$.

Proof. Suppose for contradiction that for some $u \in \kappa_2$, $V(P_u) \cap S \neq \varnothing$. Let q be the vertex in $V(P_u) \cap S$ such that $\mathrm{dist}_{P_u}(b_u, q)$ is minimum. Let P_1 be the subpath of P_u between b_u and q. Note that P_1 is an S-avoiding path of length at most r. Since $\{u, z\} \subseteq \kappa_1$, G has an S-avoiding path P_2 between $\psi_z^{-1} \circ \psi_u(b_u)$ and q having length at most that of P_1. By substituting P_1 with P_2 from P_u, we obtain a walk of G between $\psi_z^{-1} \circ \psi_u(b_u) \in B_z$ and c_u having length at most r, contradicting the assumption that $\mathrm{dist}_G(B_z, C_u) > r$. ◁

Since L is distance-r' independent in $G - S$ and $2r \leqslant r'$, by Claim 8, $c_u \neq c_{u'}$ for distinct $u, u' \in \kappa_2$. Since $|\kappa_2| \geqslant d \cdot (f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{dual}}(\lfloor r/2 \rfloor, d, \varepsilon)^{\varepsilon} \cdot k^{2\varepsilon} + s + 1)$ and every element in I contains at most d vertices, there is a set $\kappa_3 \subseteq \kappa_2$ of size at least $f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{dual}}(\lfloor r/2 \rfloor, d, \varepsilon)^{\varepsilon} \cdot k^{2\varepsilon} + s + 1$ such that $C_u \neq C_{u'}$ for all distinct $u, u' \in \kappa_3$.

Let κ_3' be the set of vertices $u \in \kappa_3$ with $\mathrm{dist}_G(C_u, S) \leqslant \lfloor r/2 \rfloor$. Since I is a $(p, r, \mathcal{F})$-packing of A in G, for all distinct $u, u' \in \kappa_3$, $\mathrm{dist}_G(C_u, C_{u'}) > r$. Thus, we deduce that $|\kappa_3'| \leqslant |S| \leqslant s$. Let $\kappa_4 := \kappa_3 \backslash \kappa_3'$. Note that $|\kappa_4| \geqslant f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{dual}}(\lfloor r/2 \rfloor, d, \varepsilon)^{\varepsilon} \cdot k^{2\varepsilon} + 1$.

We now show that κ_4 contains distinct vertices v and v' with $\mathrm{dist}_{G-S}(v, v') \leqslant r'$. For each $u \in \kappa_4$, since $G^p[C_u]$ is isomorphic to a graph in $\mathcal{F}$ and X_{cl} is a $(p, \lfloor r/2 \rfloor, \mathcal{F})$-cover of A in G, G has a path R_u of length at most $\lfloor r/2 \rfloor$ between some $y_u \in C_u$ and $x_u \in X_{\mathrm{cl}}$. Since $u \notin \kappa_3'$, $V(R_u) \cap S = \varnothing$. Since $G^p[C_u]$ is isomorphic to a connected graph in $\mathcal{F}$, G has a path Q_u of length at most $p(d-1)$ between c_u and y_u. More specifically, Q_u is a concatenation of $Q_u^1, \ldots, Q_u^{t'}$ for $t' \leqslant d-1$ such that for each $i \in [t']$, the length of Q_u^i is at most p and the ends of Q_u^i are in C_u. Since $p \leqslant 2\lfloor r/2 \rfloor + 1$, for some $j \in [t']$, if $V(Q_u^j) \cap S \neq \varnothing$, then $\mathrm{dist}_G(C_u, S) \leqslant \lfloor r/2 \rfloor$, contradicting that $u \notin \kappa_3'$. Thus, $V(Q_u) \cap S = \varnothing$. By Claim 8, $V(P_u) \cap S = \varnothing$. Since $(G-S)^p[B_u^*]$ is connected and $|B_u^*| \leqslant d$, $G - S$ has a path O_u of length at most $p(d-1)$ between u and b_u. By concatenating O_u, P_u, Q_u, and R_u, we obtain a walk of $G - S$ between u and x_u having length at most

$$|E(O_u)| + |E(P_u)| + |E(Q_u)| + |E(R_u)| \leqslant p(d-1) + r + p(d-1) + \lfloor r/2 \rfloor \leqslant \lfloor r'/2 \rfloor.$$

Let W_u be a path of $G - S$ between u and x_u consisting of edges of the walk. Let w_u be the vertex in $V(W_u) \cap X_{\mathrm{cl}}$ such that $\mathrm{dist}_{W_u}(u, w_u)$ is minimum. Such w_u exists, because $x_u \in X_{\mathrm{cl}}$. Note that the subpath of W_u between u and w_u is an X_{cl}-avoiding path of length at most $\lfloor r'/2 \rfloor$. Thus, w_u is contained in $M_{r'}^G(u, X_{\mathrm{cl}})$. Since $\{u, z\} \subseteq \kappa_1 \subseteq \lambda$ where λ is an equivalence class of $\sim$, $M_{r'}^G(u, X_{\mathrm{cl}})$ and $M_{r'}^G(z, X_{\mathrm{cl}})$ are same. Therefore, $w_u \in M_{r'}^G(z, X_{\mathrm{cl}})$.

Since $|\kappa_4| \geqslant f_{\mathrm{cl}}(r', \varepsilon) \cdot f_{\mathrm{dual}}(\lfloor r/2 \rfloor, d, \varepsilon)^{\varepsilon} \cdot k^{2\varepsilon} + 1 \geqslant |M_{r'}^G(z, X_{\mathrm{cl}})| + 1$, by the pigeonhole principle, there are distinct $v, v' \in \kappa_4$ with $w_v = w_{v'}$. By concatenating W_v and $W_{v'}$, we obtain a walk of $G - S$ between v and v' having length at most r', contradicting the assumption that L is distance-r' independent in $G - S$. Therefore, there are a vertex $z' \in \kappa_1 \backslash \{z\}$ and a set $B_{z'} \subseteq A \backslash \{z\}$ such that $z' \in B_{z'}$ and $(I \backslash \{B_z\}) \cup \{B_{z'}\}$ is a $(p, r, \mathcal{F})$-packing of $A \backslash \{z\}$ in G having the same size as I. We conclude the proof by scaling ε to ε / C. ◀

By recursively applying Proposition 4.1 and taking an $(r+1)$-path closure of the resulting set Z, we can construct an almost linear kernel for Annotated $(p, r, \mathcal{F})$-Packing as follows.

▶ **Theorem 4.3.** *For every nowhere dense class $\mathcal{C}$ of graphs, there is a function $f_{\mathrm{pck}} : \mathbb{N} \times \mathbb{N} \times \mathbb{R}_+ \to \mathbb{N}$ such that for every nonempty family $\mathcal{F}$ of connected graphs with at most d vertices, $p, r \in \mathbb{N}$ with $p \leqslant 2\lfloor r/2 \rfloor + 1$, and $\varepsilon > 0$, there is a polynomial-time algorithm that given a graph $G \in \mathcal{C}$, $A \subseteq V(G)$, and $k \in \mathbb{N}$, either correctly decides that $\alpha_{p,r}^{\mathcal{F}}(G, A) > k$, or outputs sets $Y \subseteq V(G)$ of size at most $f_{\mathrm{pck}}(r, d, \varepsilon) \cdot k^{1+\varepsilon}$ and $Z \subseteq A \cap Y$ such that $\alpha_{p,r}^{\mathcal{F}}(G, A) \geqslant k$ if and only if $\alpha_{p,r}^{\mathcal{F}}(G[Y], Z) \geqslant k$.*

To prove Theorem 1.2, we first apply the kernel in Theorem 4.3 and attach a $(p, \mathcal{F})$-critical graph to the resulting instance of this kernel. The way is similar to that of the proof of Theorem 1.1, but slightly different.

Sketch of the proof of Theorem 1.2. The cases where either $r \leqslant 1$ or $p = 0$ are relatively easy to deal with. Thus, in this sketch, we assume that $r \geqslant 2$ and $p \geqslant 1$. Let d be the maximum order of a graph in $\mathcal{F}$. By Lemma 3.4, one can find in polynomial time a $(p, \mathcal{F})$-critical graph H having at most $d(dp+1)/2$ vertices. Let $p' := \lfloor p/2 \rfloor$ and x be a vertex of H. We construct a graph G' as follows: take the disjoint union of $G[Y]$ and H, add a new vertex h, for each $v \in Y \backslash Z$, connect h and v by a path of length $\lfloor r/2 \rfloor$, and for each $v \in N_H^{p'}[x]$, connect h and v by a path of length $\lceil r/2 \rceil$. We can show that the resulting graph G' is the desired one by Lemma 3.5. ◀

References

1. Faisal N. Abu-Khzam. A kernelization algorithm for d-hitting set. *J. Comput. System Sci.*, 76(7):524–531, 2010. doi:10.1016/j.jcss.2009.09.002.
2. Hans Adler and Isolde Adler. Interpreting nowhere dense graph classes as a classical notion of model theory. *European J. Combin.*, 36:322–330, 2014. doi:10.1016/j.ejc.2013.06.048.
3. Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004. doi:10.1145/990308.990309.
4. Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009. doi:10.1007/s00453-008-9204-0.
5. Manuel Aprile, Matthew Drescher, Samuel Fiorini, and Tony Huynh. A tight approximation algorithm for the cluster vertex deletion problem. *Math. Program.*, 197(2):1069–1091, 2023. doi:10.1007/s10107-021-01744-w.
6. Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) kernelization. *J. ACM*, 63(5):Art. 44, 69, 2016. doi:10.1145/2973749.
7. Flavia Bonomo-Braberman, Julliano R. Nascimento, Fabiano S. Oliveira, Uéverton S. Souza, and Jayme L. Szwarcfiter. Linear-time algorithms for eliminating claws in graphs. In *Computing and combinatorics*, volume 12273 of *Lecture Notes in Comput. Sci.*, pages 14–26. Springer, Cham, 2020.
8. H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete Comput. Geom.*, 14(4):463–479, 1995. ACM Symposium on Computational Geometry. doi:10.1007/BF02570718.
9. Santiago Canales, Gregorio Hernández, Mafalda Martins, and Inês Matos. Distance domination, guarding and covering of maximal outerplanar graphs. *Discrete Appl. Math.*, 181:41–49, 2015. doi:10.1016/j.dam.2014.08.040.
10. Holger Dell and Dániel Marx. Kernelization of packing problems. In *23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 68–81. ACM, New York, 2012.
11. Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005. doi:10.1145/1101821.1101823.
12. Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, fifth edition, 2018. Paperback edition of [MR3644391].
13. Frederic Dorn. Dynamic programming and fast matrix multiplication. In *14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006*, volume 4168 of *Lecture Notes in Comput. Sci.*, pages 280–291. Springer, Berlin, 2006. doi:10.1007/11841036_27.
14. Rod 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.
15. Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness. II. On completeness for $W[1]$. *Theoret. Comput. Sci.*, 141(1-2):109–131, 1995. doi:10.1016/0304-3975(94)00097-3.
16. Rodney G. Downey and Michael R. Fellows. *Fundamentals of parameterized complexity*. Texts in Computer Science. Springer, London, 2013. doi:10.1007/978-1-4471-5559-1.
17. Pål Grønås Drange, Markus Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. In *33rd Symposium on Theoretical Aspects of Computer Science*, volume 47 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 31, 14. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2016. doi:10.4230/LIPIcs.STACS.2016.31.
18. Zdeněk Dvořák. Constant-factor approximation of the domination number in sparse graphs. *European J. Combin.*, 34(5):833–840, 2013. doi:10.1016/j.ejc.2012.12.004.
19. Jack Edmonds. Paths, trees, and flowers. *Canadian J. Math.*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

20 Kord Eickmeyer, Archontia C. Giannopoulou, Stephan Kreutzer, O-joung Kwon, Michał Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Neighborhood complexity and kernelization for nowhere dense classes of graphs. In *44th International Colloquium on Automata, Languages, and Programming*, volume 80 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 63, 14. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2017. doi:10.4230/LIPIcs.ICALP.2017.63.

21 Guy Even, Dror Rawitz, and Shimon Shahar. Hitting sets when the VC-dimension is small. *Inform. Process. Lett.*, 95(2):358–362, 2005. doi:10.1016/j.ipl.2005.03.010.

22 Grzegorz Fabiański, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Progressive algorithms for domination and independence. In *36th International Symposium on Theoretical Aspects of Computer Science*, volume 126 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 27, 16. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2019.

23 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (connected) dominating set on H-minor-free graphs. In *23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–92. ACM, New York, 2012.

24 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Kernels for (connected) dominating set on graphs with excluded topological minors. *ACM Trans. Algorithms*, 14(1):Art. 6, 31, 2018. doi:10.1145/3155298.

25 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. *SIAM J. Comput.*, 49(6):1397–1422, 2020. doi:10.1137/16M1080264.

26 Fedor V. Fomin and Dimitrios M. Thilikos. Fast parameterized algorithms for graphs on surfaces: linear kernel and exponential speed-up. In *Automata, languages and programming*, volume 3142 of *Lecture Notes in Comput. Sci.*, pages 581–592. Springer, Berlin, 2004. doi:10.1007/978-3-540-27836-8_50.

27 Fedor V. Fomin and Dimitrios M. Thilikos. Dominating sets in planar graphs: branch-width and exponential speed-up. *SIAM J. Comput.*, 36(2):281–309, 2006. doi:10.1137/S0097539702419649.

28 Jakub Gajarský, Petr Hliněný, Jan Obdržálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. *J. Comput. System Sci.*, 84:219–242, 2017. doi:10.1016/j.jcss.2016.09.002.

29 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):Art. 17, 32, 2017. doi:10.1145/3051095.

30 Shai Gutner. Polynomial kernels and faster algorithms for the dominating set problem on graphs with an excluded minor. In *Parameterized and exact computation*, volume 5917 of *Lecture Notes in Comput. Sci.*, pages 246–257. Springer, Berlin, 2009. doi:10.1007/978-3-642-11269-0_20.

31 Iyad Kanj, Michael J. Pelsmajer, Marcus Schaefer, and Ge Xia. On the induced matching problem. *J. Comput. System Sci.*, 77(6):1058–1070, 2011. doi:10.1016/j.jcss.2010.09.001.

32 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Trans. Algorithms*, 12(2):Art. 21, 41, 2016. doi:10.1145/2797140.

33 Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial kernels and wideness properties of nowhere dense graph classes. *ACM Trans. Algorithms*, 15(2):Art. 24, 19, 2019. doi:10.1145/3274652.

34 Jiří Matoušek. *Lectures on discrete geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2002. doi:10.1007/978-1-4613-0039-7.

35 Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem. *Discrete Appl. Math.*, 157(4):715–727, 2009. doi:10.1016/j.dam.2008.07.011.

36 James Nastos and Yong Gao. Bounded search tree algorithms for parametrized cograph deletion: efficient branching rules by exploiting structures of special graph classes. *Discrete Math. Algorithms Appl.*, 4(1):1250008, 23, 2012. doi:10.1142/S1793830912500085.

37 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion. I. Decompositions. *European J. Combin.*, 29(3):760–776, 2008. doi:10.1016/j.ejc.2006.07.013.

38 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Trans. Algorithms*, 9(1):Art. 11, 23, 2012. doi:10.1145/2390176.2390187.

39 Michał Pilipczuk and Sebastian Siebertz. Kernelization and approximation of distance-r independent sets on nowhere dense graphs. *European J. Combin.*, 94:103309, 19, 2021. doi:10.1016/j.ejc.2021.103309.

40 J. A. Telle and Y. Villanger. FPT algorithms for domination in sparse graphs and beyond. *Theoret. Comput. Sci.*, 770:62–68, 2019. doi:10.1016/j.tcs.2018.10.030.

41 Dekel Tsur. Faster parameterized algorithm for cluster vertex deletion. *Theory Comput. Syst.*, 65(2):323–343, 2021. doi:10.1007/s00224-020-10005-w.

Geometric TSP on Sets

Henk Alkema
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Mark de Berg
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Abstract

In One-of-a-Set TSP, also known as the Generalised TSP, the input is a collection $\mathcal{P} := \{P_1, ..., P_r\}$ of sets in a metric space and the goal is to compute a minimum-length tour that visits one element from each set.

In the Euclidean variant of this problem, each P_i is a set of points in $\mathbb{R}^d$ that is contained in a given hypercube H_i. We investigate how the complexity of Euclidean One-of-a-Set TSP depends on λ, the ply of the set $\mathcal{H} := \{H_1, ..., H_r\}$ of hypercubes (The ply is the smallest λ such that every point in $\mathbb{R}^d$ is in at most λ of the hypercubes). Furthermore, we show that the problem can be solved in $2^{O(\lambda^{1/d} n^{1-1/d})}$ time, where $n := \sum_{i=1}^{r} |P_i|$ is the total number of points. Finally, we show that the problem cannot be solved in $2^{o(n)}$ time when $\lambda = \Theta(n)$, unless the Exponential Time Hypothesis (ETH) fails.

In Rectilinear One-of-a-Cube TSP, the input is a set $\mathcal{H}$ of hypercubes in $\mathbb{R}^d$ and the goal is to compute a minimum-length rectilinear tour that visits every hypercube. We show that the problem can be solved in $2^{O(\lambda^{1/d} n^{1-1/d} \log n)}$ time, where n is the number of hypercubes.

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

Keywords and phrases Euclidean TSP, TSP on Sets, Rectilinear TSP, TSP on Neighbourhoods

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.6

Funding The work in this paper is supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.

1 Introduction

In the Traveling Salesman Problem we are given an edge-weighted complete graph and the goal is to compute a tour, i.e., a simple cycle visiting all nodes, of minimum total weight. The Traveling Salesman Problem is among the most famous problems in computer science and combinatorial optimization. One variation is the Euclidean TSP. In Euclidean TSP the input is a set P of n points in $\mathbb{R}^d$, and the goal is to compute a minimum-length tour visiting each point. This problem was proven to be NP-hard in the 1970s [6, 15]. However, unlike the general (metric) version, Euclidean TSP in the plane can be solved in *subexponential* time, i.e., in time $2^{o(n)}$. Both Kann [10] and Hwang et al. [8] have given algorithms with $n^{O(\sqrt{n})}$ running time. Smith and Wormald [16] gave a subexponential algorithm that works in any (fixed) dimension d, taking $n^{O(n^{1-1/d})}$ time. Recently De Berg et al. [3] improved this to $2^{O(n^{1-1/d})}$, which is tight up to constant factors in the exponent, under the Exponential-Time Hypothesis (ETH) [9].

Meanwhile, generalised versions of the Traveling Salesman Problem have also been studied. One popular example is the One-of-a-Set TSP, also known as Generalised TSP or Group TSP. Here, the n nodes of the graph are partitioned into sets V_i and the goal is to compute a tour of minimal weight visiting at least (or exactly) one node of every set. The One-of-a-Set TSP has been studied extensively, see for example the survey by Gutin and Punnen [7]. In 2008, Dror and Orlin showed that even if the vertices and their distances correspond to locations in $\mathbb{R}^d$ and their Euclidean distances, and every set V_i contains only two vertices, the problem is still APX-hard, i.e., there exists no PTAS unless P = NP [5].

One generalisation of Euclidean TSP is TSP with Neighbourhoods. Here, we are given a set of neighbourhoods in $\mathbb{R}^d$ – the shape of these neighbourhoods depends on the variant studied – and the goal is to find a shortest Euclidean tour visiting all neighbourhoods. As it is a generalised version of an APX-hard problem it is APX-hard itself [4]. Since then, many variants which place restrictions on the shape of the neighbourhoods have been shown to have a PTAS. This includes disjoint fat convex regions [4], pairwise-disjoint connected regions of any shape [14], arbitrary disjoint fat regions [13], and weakly disjoint neighbourhoods [2].

In this paper, we focus on two geometric variations of One-of-a-Set TSP. For the first variation, Euclidean One-of-a-Set TSP, most research has been focused on the so-called *grid cluster* variant, introduced by Bhattacharya et al. [1]. In this variant, a partition is specified by the cells of the integer 1×1 grid (on the Euclidean plane); from every non-empty cell, exactly one point needs to be visited. Khachay and Neznakhina showed that a PTAS exists if there are many ($O(\log n)$) or few ($n - O(\log n)$) non-empty cells [11], and that if the grid has fixed height or width, a solution can be found in polynomial time [12].

For the second variation, Rectilinear One-of-a-Cube TSP, instead of discrete sets V_i we are given a set of (hyper)cubes H_i. The goal is to find the shortest rectilinear tour which visits at least one point of every H_i. Thus this is a variant of TSP with Neighbourhoods where the neighborhoods are hypercubes.

Our contribution. We investigate the complexity of Euclidean One-of-a-Set TSP and Rectilinear One-of-a-Cube TSP. For Euclidean One-of-a-Set TSP, let $\mathcal{H}$ be a set of hypercubes $H_1, ..., H_r$. Let $\mathcal{P}$ be a family of sets of points $P_1, ..., P_r$ with $P_i \subset H_i$ and $|P_i| \leq k$ for all i. We will use P to denote $\cup_i P_i$. Our objective is to find a shortest tour $T = (p_1, ..., p_n)$ such that for every P_i there exists a $p \in P_i$ such that $p \in T$. Let λ be the *ply* of the given hypercubes, i.e., the smallest number such that every point in $\mathbb{R}^d$ is in at most λ of the hypercubes.

Intuitively, one would expect that the complexity of Euclidean One-of-a-Set TSP depends on how well separated the sets P_i are. To formalize this intuition, we investigate the dependency of its complexity on the ply of $\mathcal{H}$. We present an algorithm that runs in $2^{O(\lambda^{1/d} n^{1-1/d})}$ time, which is based on a recent algorithm by De Berg et al. [3]. Note that for $\lambda = 1$ this matches the ETH-tight running time for Euclidean TSP. For $\lambda = n$, however, the running time becomes $2^{O(n)}$, so it is no longer sub-exponential. We show this is unavoidable (assuming ETH) by proving that Euclidean One-of-a-Set TSP in $\mathbb{R}^2$ cannot be solved in $2^{o(n)}$ for $\lambda = n$. Finally, we show that we instead of using the ply of a set of hypercubes covering the P_i, one can use the ply of a set of more generic objects covering the P_i.

For Rectilinear One-of-a-Cube TSP, let $\mathcal{H}$ be a set $\{H_1, ..., H_n\}$ of hypercubes in $\mathbb{R}^d$. Let λ be the *ply* of these hypercubes. Our objective is to find a shortest rectilinear tour $T = (p_1, ..., p_n)$ such that for every H_i there exists a $p \in H_i$ such that $p \in T$. For this case we present an algorithm running in $2^{O(\lambda^{1/d} n^{1-1/d}) \log n}$ time.

2 A subexponential algorithm for Euclidean One-of-a-Set TSP

We start by giving an overview of our algorithm. The problem it solves is the more generic Euclidean One-of-a-Set Path Cover problem. In this problem, we are given a collection of point sets P_i and a set of *boundary points* B, where $|B| \geq 2$ is even. The goal is for every possible matching on B to find the shortest collection of paths that (i) have the so-called *Packing Property*, (ii) adhere to the matching and (iii) visit one point of every P_i. The Packing Property, which is known to hold for the set of edges of an optimal TSP tour, intuitively states that an optimal tour cannot contain many long edges close together; the precise definition is not important for this paper. Note that this property also holds for an

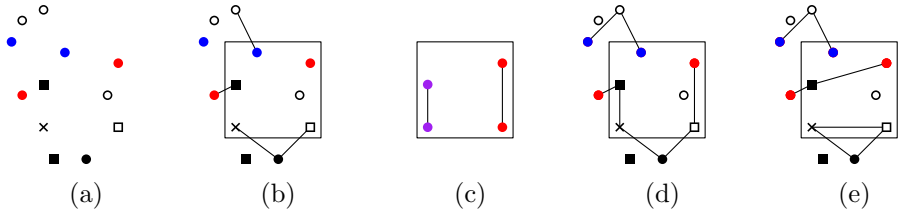


Figure 1 An example of the Euclidean One-of-a-Set TSP algorithm. (a) The given problem. In red and blue, the four boundary points, and the corresponding matching we will be using. In black, the five given point sets (circles, disks, squares, filled squares, and crosses). (b) We find a good separator σ, and guess how it is crossed. (c) The resulting subproblem for the points inside σ, and one possible matching. Note that the circle inside σ has been removed, as we guessed we visit a circle outside σ. (d) The answer after combining two answers of the subproblems. (e) Note that not every combination of matchings leads to a valid answer.

optimal Euclidean One-of-a-Set TSP tour T_{opt}, as it is obviously also an optimal tour for the Euclidean TSP instance obtained by taking only a single point from each P_i into account, namely the point visited by T_{opt}.

Our algorithm broadly works in the following way; see Figure 1. We define a *separator* to be the boundary of an axis-aligned hypercube. First, we find a separator σ that is crossed only a few times by the optimal collection of paths, and splits only few of the point sets P_i. Then, we "guess" how the solution crosses σ, by iterating over all possibilities. By doing so, we create two subproblems: one for the points inside σ, and one for the points outside σ. However, these two subproblems are not completely independent yet: first, for every point set P_i which contains at least one point inside σ and one outside σ, we need to "guess" on which side of σ we visit a point of P_i. After solving all versions of both subproblems recursively, the so-called rank-based approach is used to efficiently find the correct combination of matchings on both sides resulting in the shortest overall valid answer.

A good separator. Our algorithm will need a so-called distance-based separator, similar to the distance-based separator introduced by De Berg et al. [3]. The properties they proved for their separator are not quite sufficient for us, though, so below we present a stronger version of their separator result. Before we can state our result, we need to introduce some terminology and notation from their paper. We denote the region of all points in $\mathbb{R}^d$ inside or on a separator σ by σ_{in}, and the region of all points in $\mathbb{R}^d$ strictly outside σ by σ_{out}. The *size* of a separator σ, denoted by $\mathit{size}(\sigma)$, is defined to be its edge length. For a separator σ and a scaling factor $t > 0$, we define $t\sigma$ to be the separator obtained by scaling σ by a factor t with respect to its center. In other words, $t\sigma$ is the separator whose center is the same as the center of σ and with $\mathit{size}(t\sigma) = t \cdot \mathit{size}(\sigma)$. Note that a separator σ induces a partition of the given point set P into two subsets, namely $P \cap \sigma_{\text{in}}$ and $P \cap \sigma_{\text{out}}$. A separator is *balanced* with respect to a set $Q \subseteq P$ if $\max(|Q \cap \sigma_{\text{in}}|, |Q \cap \sigma_{\text{out}}|) \leq \frac{4^d}{4^d+1} n$. If a separator is balanced with respect to P itself, we call it a balanced separator.

Our separator is chosen such that there are only few points close to it. To quantify this, let the *relative distance* from a point p to σ, denoted by $\mathit{rdist}(p, \sigma)$, be defined as follows:

$$\mathit{rdist}(p, \sigma) := d_\infty(p, \sigma)/\mathit{size}(\sigma),$$

where $d_\infty(p, \sigma)$ denotes the shortest ℓ_∞-distance between p and any point on σ. Recall that σ is the boundary of a hypercube; hence, all points in the interior of σ_{in} have a nonzero relative distance to σ. Note that if t is the scaling factor such that $p \in t\sigma$, then $\mathit{rdist}(p, \sigma) = |1 - t|/2$. For integers i define

$$P(i,\sigma) := \{p \in P : rdist(p,\sigma) \leq 2^i/n^{1/d}\}.$$

Note that the smaller i is, the closer to σ the points in $P(i,\sigma)$ are required to be. De Berg et al. choose σ such that the size of the sets $P(i,\sigma)$ decrease rapidly as i decreases. Our generalised theorem also allows for some control over the number of P_i which are *split* by σ, i.e., the P_i of which at least one point of P_i is in σ_{in} and at least one point of P_i is in σ_{out}.

▶ **Theorem 1.** *Let $\mathcal{P} = \{P_1, ..., P_r\}$ be a collection of point sets in $\mathbb{R}^d$, and let $Q \subseteq P$. Let n be the total number of points in P. Then there is a separator σ that is balanced with respect to Q and such that*

$$|P(i,\sigma)| = \begin{cases} O((3/2)^i n^{1-1/d}) & \text{for all } i < 0 \\ O(4^i n^{1-1/d}) & \text{for all } i \geq 0. \end{cases}$$

Furthermore, σ splits at most $\lambda^{1/d} n^{1-1/d}$ of the point sets P_i. Moreover, such a separator can be found in $O(n^{d+1})$ time.

Proof. The proof is analogous to the proof of Theorem 1 and Corollary 3 in the paper by De Berg et al. [3], with one major difference. Instead of the weight function $w_p(t)$, we use the weight function

$$w_p^*(t) := \frac{\mathbf{1}\{t\sigma^* \text{ intersects } H_{i(p)}\}}{size(H_{i(p)})} + w_p(t),$$

where σ^* is the smallest balanced separator, which we assume w.l.o.g. has size 1 (as in the original proof), $i(p)$ denotes the i such that $p \in P_i$, and $\mathbf{1}\{bool\}$ denotes the indicator function, which is 1 if *bool* is true, and 0 otherwise. Since $\int_0^3 w_p(t) = O(1)$, we have $\int_0^3 w_p^*(t) = O(1)$ as well, because $\int_0^3 \mathbf{1}\{t\sigma^* \text{ intersects } H_{i(p)}\} \leq size(H_{i(p)})$. Hence, we can find a t^* such that $\sum_p w_p^*(t^*) = O(n)$. Therefore, we can use this t^* to prove the bounds on $|P(i,\sigma)|$ analogously to the original proofs.

It remains to prove the bound on the number of P_i split by $t^*\sigma^*$. We get

$$\sum_i \frac{\mathbf{1}\{t^*\sigma^* \text{ intersects } H_i\}}{size(H_i)} \leq \sum_p \frac{\mathbf{1}\{t^*\sigma^* \text{ intersects } H_{i(p)}\}}{size(H_{i(p)})} < \sum_p w_p^*(t) = O(n).$$

Therefore, for any L, at most $O(nL)$ different H_i of size at most L intersect $t\sigma^*$. Furthermore, an H_i of size at least L intersecting σ covers at least L^{d-1} of the $(d-1)$-dimensional volume of σ, which is $2d = O(1)$. (Recall that σ is defined as the boundary of a hypercube, which is a $(d-1)$-dimensional object, and that we consider d to be fixed.) See Figure 2. Since the H_i have a ply of λ, there can be at most λ/L^{d-1} of these. Hence, for any L, at most $O(nL + \lambda/L^{d-1})$ hypercubes intersect σ. Specifically, by taking $L = \lambda^{1/d} n^{-1/d}$, we conclude that σ intersects $O(\lambda^{1/d} n^{1-1/d})$ of the H_i. Finally, we note that the number of P_i split by σ is bounded by the number of H_i intersected by σ, finishing our proof. ◀

We will now use this distance-based separator theorem to present an efficient algorithm for Euclidean One-of-a-Set TSP.

Let $S(P) := \{pq : (p,q) \in P \times P\}$ be the set of all line segments defined by P. Since we wish to guess how σ is crossed by the optimal answer, and we know that the edges of the answer have the packing property, we are interested in the following set:

$$\mathcal{C}(\sigma, P) := \{S \subseteq S(P) : S \text{ has the packing property and all segments in } S \text{ cross } \sigma\}.$$

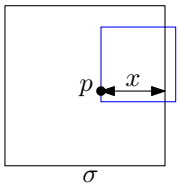


Figure 2 Example for the proof of Theorem 1. The point p is at distance x from σ. Hence, any square H (in blue) that contains both p and a point outside σ covers at least x of the total length of the edges of σ. For arbitrary d, any d-dimensional hypercube H that contains both p and a point outside σ covers at least x^{d-1} of the $(d-1)$-dimensional volume of σ. (Recall that σ is defined to be the $(d-1)$-dimensional boundary of a hypercube, so σ does not include the region enclosed by it.)

Our main separator theorem, presented next, states that we can find a separator σ that is balanced, splits few P_i, and is such that the sets in $\mathcal{C}(\sigma, P)$, as well as the collection $\mathcal{C}(\sigma, \mathcal{P})$ itself, are small. Since the packing property is hard to test, we will not enumerate $\mathcal{C}(\sigma, P)$ but a slightly larger collection of candidate sets, which we denote by $\mathcal{C}'(\sigma, P)$.

▶ **Theorem 2.** *Let $\mathcal{P} = \{P_1, ..., P_r\}$ be a family of point sets in $\mathbb{R}^d$, and let $\mathcal{H} = \{H_1, ..., H_r\}$ be a set of hypercubes such that $P_i \subset H_i$ for every i. Let $Q \subseteq P$, where $P = P_1 \cup \cdots \cup P_r$. Then there exists a separator σ with the following properties:*

1. *σ is balanced with respect to Q.*
2. *Each candidate set $S \in \mathcal{C}'(\sigma, P)$ contains $O(n^{1-1/d})$ segments.*
3. *$\mathcal{C}(\sigma, P) \subseteq \mathcal{C}'(\sigma, P)$ and $|\mathcal{C}'(\sigma, P)| = 2^{O(n^{1-1/d})}$.*
4. *σ splits $O(\lambda^{\frac{1}{d}} n^{1-1/d})$ of the sets P_i, where λ is the ply of $\mathcal{H}$.*

Moreover, σ and the collection $\mathcal{C}'(\sigma, P)$ can be computed in $2^{O(n^{1-1/d})}$ time.

Proof. The separator chosen is the one found by applying Theorem 1. Hence, the proof of properties 1-3 is analogous to that of the original paper by De Berg et al. Property 4 is directly implied by Theorem 1, as well. ◀

The algorithm. Our adapted algorithm contains four changes compared to the original:

- We choose our separator σ using Theorem 2 instead of the equivalent from the original paper. Note that this does not impact the running time.
- Candidate sets of which the endpoints of the edges contain more than one point of any set P_i can be ignored. Note that this can be easily checked.
- When "guessing" the correct candidate set, for every point set P_i split by σ we also guess whether a point of P_i in σ_{in} or a point in σ_{out} is used. If we guess that we will visit a point in $P_i \cap \sigma_{\text{in}}$, then we can ignore the points in $P_i \cap \sigma_{\text{out}}$ for the recursive call outside σ, and vice versa. Note that for every P_i that contains a boundary point this choice (if applicable) is implied by the location of the boundary point. Furthermore, once we have chosen a point from a set P_i as one of our boundary points, then we can remove all other points from P_i from further consideration. This way, the subproblems generated remain independent, while ensuring that exactly one point of every P_i is visited.
- In the initial call, the original algorithm turns the problem into a Euclidean Path Cover problem by duplicating point p_1 and taking the boundary set $B = (p_1, p_1')$. In other words, it simply searches for a path from p_1 to p_1 through all other points. In our case, we guess which p in P_1 is used in the optimal tour. Then, we remove all other points in P_1 and duplicate p as in the original algorithm.

This brings us to our main theorem for this section.

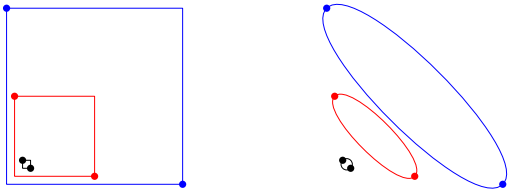


Figure 3 On the left, an example of a point set whose hypercubes generate a high ply. The pattern can be repeated to get a ply of $\Theta(n)$. However, the separator we find will not split $\Theta(n)$ point sets. On the right, the same point set covered by α-fat objects. Here, we have a ply of only 1.

▶ **Theorem 3.** *Let $\mathcal{P} = \{P_1, .., P_r\}$ be a collection of point set in $\mathbb{R}^d$ with n points in total. Let $H_1, ..., H_r$ be hypercubes with ply λ such that $P_i \subset H_i$ for all i. Then* Euclidean One-of-a-Set TSP *on $\mathcal{P}$ can be solved in $2^{O(\lambda^{\frac{1}{d}} n^{1-1/d})}$ time.*

For the full proof, see Appendix A. It follows the proof of the original algorithm [3] almost verbatim; the main difference is that we need to take the dependency on λ into account.

An improved analysis of the running time. So far, we have used the ply of hypercubes covering the point sets to bound the running time of our algorithm. (Note that the algorithm itself does not use the hypercubes, we only used their ply in the analysis to quantify how separated the sets are.) However, in some cases, these hypercubes can have a high ply, even though they are still fairly well separable. Thus the analysis may be overly pessimistic. We show that this is indeed the case by replacing the hypercubes by so-called *α-fat objects*, whose ply can be much smaller than the ply of the hypercubes. let $0 < \alpha < 1$ be arbitrary but fixed. We say a connected closed set of points $O \subseteq \mathbb{R}^d$ is an α-fat object if and only if for every ball $B \subset \mathbb{R}^d$ whose center lies in O we have that (i) O fully lies in B or (ii) at least a fraction α of the volume of B is covered by O. See Figure 3 for an example showing how α-fat objects can have significantly lower ply than hypercubes.

It remains to show that with this new λ our running time of $2^{O(\lambda^{1/d} n^{1-1/d})}$ is still accurate. Note that we use the fact that the objects containing the sets P_i are hypercubes only once, namely during the proof of Theorem 1, where we bound the number of H_i split by σ to $2^{O(\lambda^{1/d} n^{1-1/d})}$. We will now prove that a similar bound holds for arbitrary α-fat objects.

Before we define the size of an α-fat object, we need the following observation.

▶ **Observation 4.** *Let $\alpha > 0$ be arbitrary but fixed. Let O be an α-fat object. Let $\mathcal{B}$ be the bounding box of O, i.e., the smallest box such that O lies fully in $\mathcal{B}$. Then the dimensions of $\mathcal{B}$ are within a constant factor of each other.*

Proof. Let α, O and $\mathcal{B}$ be as defined above. Let s be the largest dimension of $\mathcal{B}$. W.l.o.g., $s = 1$. Let p be a point in O. W.l.o.g., p is the origin. Note that O fully lies in $[-1, 1]^d$. Let B be the ball of radius $1/3$ centered at the origin. Note that B does not fully cover O (in fact, any radius strictly smaller than $1/2$ suffices). Hence, since O is an α-fat object, at least $\alpha c_d / 3^d$ of B (and therefore $[-1, 1]^d$) is covered by O, where c_d is the volume of a d-dimensional ball with radius 1. Since for any dimension x of $\mathcal{B}$, the volume of O inside $[-1, 1]^d$ can be bounded by x, we get that $x \geq \alpha c_d / 3^d = O(1)$, as required. ◀

We now define the size of an α-fat object O to be the largest dimension of its bounding box.

We will now show that these objects have all required properties.

▶ **Lemma 5.** *Let $\mathcal{P} = \{P_1, ..., P_r\}$ be a collection of point sets in $\mathbb{R}^d$, and let $Q \subseteq P$. Let n be the total number of points in $\mathcal{P}$. Let σ be the separator found when applying Theorem 1 with α-fat objects instead of hypercubes. Then σ splits $O(\lambda^{1/d} n^{1-1/d})$ point sets P_i.*

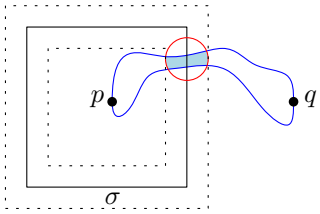


Figure 4 The dotted lines denote the square annulus A of all points with an *rdist* of at most $2^j/n^{1/d}$ to σ. In blue, the object O_i crossing σ. It has at least one of (i) a point p in σ_{in} but not in A, or (ii) a point q in σ_{out} but not in A. Therefore, if we draw a ball (in red) the width of A centered at a point where O_i coincides with σ, this circle does not fully contain O_i. Hence, the volume of O_i inside the ball (light blue), and therefore inside A, is at least α times the volume of the ball.

Proof. First, we note that the logic showing that $w_p^*(t) = O(1)$ still holds for α-fat objects. Furthermore, analogously to the original proof of Theorem 1, there are $O(\lambda^{1/d}n^{1-1/d})$ objects of size at most $\lambda^{1/d}n^{-1/d}$ intersected by σ. It remains to prove that σ splits $O(\lambda^{1/d}n^{1-1/d})$ point sets whose objects have size larger than $\lambda^{1/d}n^{-1/d}$. Note that it is sufficient to prove that $O(\lambda^{1/d}n^{-1/d})$ objects of size larger than $O(\lambda^{1/d}n^{-1/d})$ intersect σ.

Let O_i be an object of size strictly larger than $\lambda^{1/d}n^{-1/d}$ intersected by σ. Then O_i must contain a point at a distance more than $c_{\alpha,d}\lambda^{1/d}n^{-1/d}$ from σ for some constant $c_{\alpha,d}$: this distance is clearly nonzero, as O_i has a positive volume, and since α and d are fixed, the problem scales linearly.

For brevity, we write $x := c_{\alpha,d}\lambda^{1/d}n^{-1/d}$. Let B be the ball with radius x and centered at an intersection point of O_i and σ. Now, B does not fully cover O. Therefore, O covers at least a fraction α of B. Let A be the square annulus defined by $\{p \in \mathbb{R}^d : \mathit{rdist}(p, \sigma) \leq x\}$. Note that B fully in A. See Figure 4 for an example. Since B is d-dimensional, every O_i covers $\Theta(x^d)$ of the volume of A. Now, the total volume of A is smaller than $2d \cdot (1+2x)^{d-1}2x$, since each of the $2d$ facets of σ contributes $(1+2x)^{d-1}2x$ to the annulus. (This is a conservative estimate since we ignore overlap between the contributions of the facets.) Since $x < 1$, this can be further bounded by $3^{d+2}dx$. Furthermore, the volume of a d-dimensional ball with radius x is $c_d x^d$ for some constant c_d. Therefore, the maximum number of O_i intersected by σ of size strictly larger than $\lambda^{1/d}n^{-1/d}$ is bounded by

$$\lambda \frac{3^{d+2}dx}{\alpha c_d x^d} = O(\lambda x^{1-d}) = O(\lambda \cdot \lambda^{(1-d)/d} n^{-(1-d)/d}) = O(\lambda^{1/d} n^{1-1/d}),$$

as we wanted to prove. ◀

We thus obtain the following theorem.

▶ **Theorem 6.** *Let $\alpha > 0$ be arbitrary but fixed. Let $\mathcal{P} = \{P_1, .., P_r\}$ be a collection of point set in $\mathbb{R}^d$ with n points in total. Suppose there exist $O_1, ..., O_r$ be α-fat objects with ply λ such that $P_i \subset O_i$ for all i. Then Euclidean One-of-a-Set TSP on $\mathcal{P}$ can be solved in $2^{O(\lambda^{\frac{1}{d}} n^{1-1/d})}$ time.*

A lower bound on the running time when $\lambda = \Theta(n)$. In this section we show that for $\lambda = \Theta(n)$, the problem cannot be solved in subexponential time.

▶ **Theorem 7.** *Euclidean One-of-a-Set TSP in $\mathbb{R}^2$ cannot be solved in $2^{o(n)}$ time, unless ETH fails.*

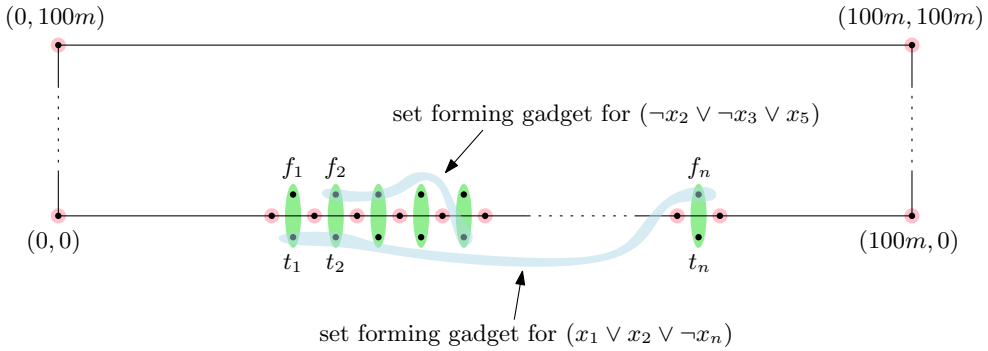


Figure 5 An example for the proof of Theorem 7. Not to scale. The points inside the small pink disks (which are the points p_i and q_i as defined in the text) must all be visited. The sets indicated by the green ellipses ensure that for each i at least one of t_i and f_i is visited. The sets indicated by the blue regions correspond to the clauses. The pink and green sets imply a lower bound on the length of the shortest tour. If this bound is tight, the shortest tour visits exactly one of each pair (t_i, f_i). Note that each such tour maps directly to an assignment of True and False to the variables. If an assignment of True and False to the variables satisfying all clauses exists, then the corresponding tour indeed visits all sets at least once.

Proof. The ETH states that 3-SAT cannot be solved in $2^{o(n)}$ time [9]. We will prove Theorem 7 by showing that if Euclidean One-of-a-Set TSP can be solved in $2^{o(n)}$ with $d = 2$ and $\lambda = \Theta(n)$, then 3-SAT can be solved in $2^{o(n)}$ time as well.

Let F be a 3-SAT formula containing clauses $C_1, ..., C_n$ over variables $x_1, ..., x_m$. Note that $m = O(n)$. We define $p_1 = (0, 0)$, $p_2 = (0, 100m)$, $p_3 = (100m, 100m)$, and $p_4 = (100m, 0)$. Furthermore, let $q_i = (50m - 2i, 0)$ for all $i = 0, ..., m$, let $t_i = (50m - 2i + 1, -1)$ and let $f_i = (50m - 2i + 1, 1)$ for all $i = 1, ..., m$. See Figure 5 for an example. Now, let $\mathcal{P}$ be the family containing the following point sets:

- For all $0 \leq i \leq 3$, one point set containing only p_i.
- For all $1 \leq i \leq m$, one point set containing t_i and f_i. This is the gadget representing the variable x_i.
- For all $1 \leq i \leq n$, one point set representing the clause C_i. Specifically, this point set should contain t_j iff C_i contains the literal x_j, and should contain f_j iff C_j contains the literal $\neg x_j$. Note that each of these point sets contains three elements, and that each point in a clause gadget coincides with a point from a variable gadget.

Now we claim that F is satisfiable if and only if $\mathcal{P}$ admits a shortest tour of length exactly $L := (398 + 2\sqrt{2})m$. Note that if this claim indeed holds, we are done; if we can solve Euclidean One-of-a-Set TSP in $2^{o(n)}$ time, then the shortest tour on $\mathcal{P}$ can be found in $2^{o(n+m)} = 2^{o(n)}$ time, and therefore we answer whether F is satisfiable in $2^{o(n)}$ time.

Note that if a shortest tour of length L exists, F is satisfiable. To satisfy F, simply set x_i to True if t_i is visited in the shortest tour, and set it to False otherwise. Since the shortest tour visits every point set corresponding to a clause, all clauses are indeed satisfied.

Next, we check that if F is satisfiable, a shortest tour of length L exists. We simply do the reverse: let $x_1, ..., x_m$ satisfy F. Then note that the tour passing through t_i if x_i is True and through f_i otherwise indeed has the required length. Finally, we still have to show the shortest tour can never be shorter than L. It is easy to see that the shortest tour must visit $q_m, p_1, p_2, p_3, p_4, q_0$ consecutively in that order, giving a length of $398m$. Connecting q_0 to q_m while passing through at least one of every pair (t_i, f_i) and through every q_i inbetween takes a total length of at least $2\sqrt{2}m$. Therefore, the shortest tour has length at least $(398 + 2\sqrt{2})m$.

In conclusion, F is satisfiable if and only if the shortest tour on $\mathcal{P}$ has length L. Therefore, if Euclidean One-of-a-Set TSP can be solved in subexponential time when $\lambda = \Theta(n)$, then so can 3-SAT. Hence, unless ETH fails, Euclidean One-of-a-Set TSP cannot be solved in subexponential time. ◀

3 Rectlinear One-of-a-Cube TSP

We continue with Rectilinear One-of-a-Cube TSP. Recall that for this setting, $\mathcal{H} := \{H_1, ..., H_n\}$ is a set of hypercubes, and λ is the ply of $\mathcal{H}$, i.e., the smallest number such that every point in $\mathbb{R}^d$ is in at most λ of the hypercubes. We now want to find a minimum-length rectilinear tour visiting all of the hypercubes H_i.

The algorithm works using the same divide-and-conquer approach as the Euclidean One-of-a-Set TSP algorithm; see the beginning of Section 2 for a more detailed description.

Properties of an optimal tour. We start by limiting the set of points and edges we need to consider. We show that there is an optimal tour using only edges from a specific set, and that these edges have the packing property.

First, we introduce some terminology. An *edge* is defined as a rectilinear line segment. A *link* between two points is any shortest path formed by at most d edges of different orientations (which always exists). We define, with slight abuse of notation, $|pq|$ to be the L_1-distance between points p and q. Note that this is also the length of a link between p and q. A *tour* is a sequence of links, where the endpoint of each link in the sequence is the starting point of the next one, and the endpoint of the last link is the startpoint of the first link. Note that the fact that we see the tour as a sequence (and not as a cycle) implies that tours have a starting point and a direction – this is solely for the purpose of the analysis.

Let q^i denote the i'th coordinate of a point q. Define C to be the set of $2^d n$ corners of the cubes in the input set $\mathcal{H}$. Let G be the *generalised Hanan grid* induced by the set C which is defined as the grid formed by drawing all axis-aligned lines through every point in the point set $C^* := \{p \in \mathbb{R}^d : \forall 1 \leq i \leq d : \exists c \in C : p^i = c^i\}$. In other words, the lines of the grid G are the intersections of $d-1$ differently oriented hyperplanes each containing the facet of one of the hypercubes.

▶ **Lemma 8.** *There exists a shortest tour on $\mathcal{H}$ which lies fully on G.*

For the full proof, see Appendix B. Intuitively, this can be done by taking any shortest tour T and "shifting" it onto the grid, bit by bit.

Given a tour T, we can reorder the hypercubes in $\mathcal{H}$ such that H_i is the i'th hypercube visited by T; ties can be broken in any way. Define p_i to be the first point where H_i is visited by T. Define $P(T) := \{p_1, ..., p_n\}$; we call the points in $P(T)$ the *entry points* of T. Note that the length of a shortest tour T that visits the points p_i in the given order equals $\sum_{1 \leq i \leq n} |p_i p_{i+1}|$, where we define $p_{n+1} := p_1$. (Recall that $|pq|$ denotes the L_1-distance from p to q.) We say a tour T is a *canonical tour* on $\mathcal{H}$ if it has the following properties:

1. T is a shortest tour on $\mathcal{H}$
2. T lies fully on G
3. Each pair of consecutive entry points in $P(T)$ is connected by a link, that is, the portion of T connecting consecutive entry points consist of at most d edges of different orientations.

▶ **Observation 9.** *For every $\mathcal{H}$, a canonical tour on $\mathcal{H}$ exists.*

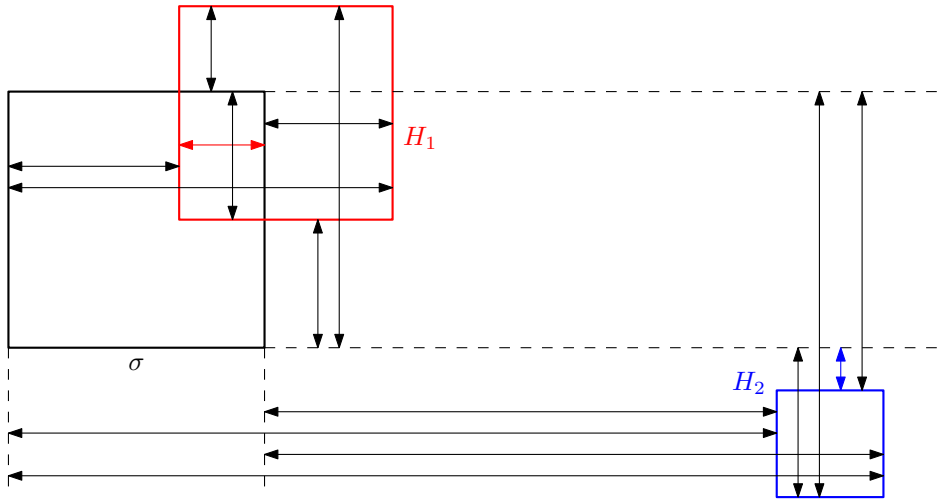


Figure 6 An example of *odrdist*. Given an H_i, in every direction e_j the four distances between one of the sides of H_i perpendicular to e_j and one of the sides of σ perpendicular to e_j are measured. The shortest of all the measured distances, scaled such that $\text{size}(\sigma) = 1$, defines the *odrdist* between the two. Note how H_1, in red, intersects σ and H_2, in blue, seems far away from σ. Yet, the *odrdist* of H_1, denoted by the red arrow, is larger than that of H_2, denoted by the blue arrow.

Proof. By Lemma 8 there exists a shortest tour T which lies fully on G, satisfying the first two properties. Now, we can create a new tour T' by creating a link between every two consecutive p_i in $P(T)$. Since T' is a shortest tour on a set of points on T, it must be a shortest tour itself as well. Furthermore, since T' visits all p_i in $P(T)$, it visits all H_i. Finally, since T lies on the generalised Hanan grid G, so do the points p_i. Furthermore, note that a link between two points on G lies on G itself. We conclude that T' has the required properties. ◀

▶ **Lemma 10.** *The edges of a canonical tour have the Packing Property.*

For the full proof, see Appendix C. Intuitively, suppose we have two long edges in the same direction (e.g. left to right) and close to each other. We can then replace these edges by two new edges – one connecting the two starting points of the removed edges and one connecting the end points – whose total length is shorter.

A good separator. As mentioned, our algorithm will be a divide-and-conquer algorithm, based on separators. Thus we need a good separator for tours on hypercubes. Our separator will again be based in the distance-based separator from [3]. It will not be sufficient to work with a distance-based separator on the corners of the hypercubes. Instead, we want to have only a few H_i with a facet close to one of the parallel facets of σ, measured in the dimension perpendicular to these facets. To be precise, for a hypercube H, let $\text{center}(H)$ be its center and $\text{size}(H)$ its edge length. Let the *one-dimensional distance* from a hypercube H to a separator σ, denoted by $oddist(H, \sigma)$ be defined as the minimum distance between any pair of parallel hyperplanes h, h' such that h contains a facet of H and h' contains a facet of σ. The *one-dimensional relative distance* from H to σ, denoted by $odrdist(H, \sigma)$ is now defined as $odrdist(H, \sigma)/size(\sigma)$. See Figure 6 for an example. For integers j define

$$P_j(\sigma) := \{H \in \mathcal{H} : 0 < odrdist(H, \sigma) \leq 2^j/n^{1/d}\}.$$

We can now prove the following theorem.

▶ **Theorem 11.** *Let $\mathcal{H}$ be a set of n hypercubes in $\mathbb{R}^d$ and let $\mathcal{I} \subseteq \mathcal{H}$. Then there is a separator σ that is balanced with respect to the corner points of $\mathcal{I}$, and such that*

$$|P_j(\sigma)| = \begin{cases} O((3/2)^j n^{1-1/d}) & \text{for all } j < 0 \\ O(4^j n^{1-1/d}) & \text{for all } 0 \leq j < \infty. \end{cases}$$

Furthermore, at most $O(\lambda^{1/d} n^{1-1/d})$ elements of $\mathcal{H}$ intersect σ. Moreover, such a separator can be found in $O(n^{d+1})$ time.

Proof. Let C be the set of corner points of the hypercubes in $\mathcal{I}$. Let σ^* be a smallest separator such that $|\sigma^*_{\text{in}} \cap C| \geq (4n)/(4^d + 1)$. As in the original proof, one can argue that for all $1 \leq t \leq 3$, the separator $t\sigma^*$ is balanced w.r.t. C. Assume w.l.o.g. that $size(\sigma^*) = 1$. Define $j_H(t)$ to be the integer such that

$$2^{j_H(t)-1}/n^{1/d} < odrdist(H, t\sigma^*) \leq 2^{j_H(t)}/n^{1/d},$$

where $j_H(t) = \infty$ if $odrdist(H, t\sigma^*) = 0$. We define the weight function as

$$w_H(t) := \frac{\mathbf{1}\{H \text{ intersects } t\sigma^*\}}{size(H)} + \begin{cases} \frac{n^{1/d}}{(3/2)^{j^i_H(t)}} & \text{if } j^i_H(t) < 0 \\ \frac{n^{1/d}}{4^{j^i_H(t)}} & \text{otherwise } 0 \leq j^i_H(t) < \infty \\ \text{undefined} & \text{otherwise.} \end{cases}$$

(Recall that $\mathbf{1}\{bool\}$ denotes the indicator function, which is 1 if *bool* is true, and 0 otherwise.) Now, for every $H \in \mathcal{H}$ we have $\int_1^3 w_H(t)dt = O(1)$, since the second part of $w_H(t)$ can be expressed as the maximum of $2d$ different versions of the weight function $w_p(t)$ of the original proof. Since H can obviously intersect $t\sigma$ only during an interval of t of size size(H), we get that $\int_1^3 w_H(t)dt = O(1)$.

Therefore, we can find a t^* such that $\sum_H w_H(t^*) = O(n)$. We claim that $t^*\sigma^*$ has the desired properties. We have already shown that it is balanced w.r.t. the corner points of $\mathcal{I}$.

Let $1 \leq i \leq d$, and let $j < 0$. Then each element in $P^i_j(\sigma)$ contributes at least $w^i_H(t^*) \leq \frac{n^{1/d}}{(3/2)^j}$ to the total weight. Therefore, there are at most $O(n/\frac{n^{1/d}}{(3/2)^j}) = O(n^{1-1/d}(3/2)^j)$ such elements, as required. The case for $0 \leq j < \infty$ can be proven analogously.

Finally, we note that at most $O(2d\lambda/(\lambda^{1/d}n^{-1/d})^{d-1}) = O(\lambda^{1/d}n^{1-1/d})$ of the H_i of size at least $\lambda^{1/d}n^{-1/d}$ can intersect σ (otherwise, somewhere, $\lambda+1$ would overlap). Furthermore, there are $O(\frac{n}{1/(\lambda^{1/d}n^{-1/d})}) = O(\lambda^{1/d}n^{1-1/d})$ of the H_i of size at most $\lambda^{1/d}n^{-1/d}$ that intersect σ, as they all contribute weight at least $1/(\lambda^{1/d}n^{-1/d})$. Therefore, there are $O(\lambda^{1/d}n^{1-1/d})$ hypercubes intersecting σ.

As in the original proof, we can argue that we can compute $t^*\sigma^*$ quickly by truncating $w^i_H(t)$. ◀

This brings us to the candidate sets. Instead of guessing how we cross the separators precisely, guessing *where* we cross the separators will suffice. For simplicity, we consider the boundary points created this way to be infinitely small hypercubes.

▶ **Theorem 12.** *Let $\mathcal{H} = \{H_1, ..., H_n\}$ be a set of n hypercubes in $\mathbb{R}^d$ and let $\mathcal{I} \subseteq \mathcal{H}$. Then there is a separator σ and a collection $C'(\sigma, \mathcal{H})$ of candidate point sets such that*

1. *σ is balanced with respect to the corner points of $\mathcal{I}$*
2. *Each candidate set $X \in C'(\sigma, H)$ contains $O(n^{1-1/d})$ points.*
3. *There exists a shortest tour T and an $X \in C'(\sigma, \mathcal{H})$ such that X is the set of locations where T crosses σ, and $|C'(\sigma, \mathcal{H})| \leq 2^{O(n^{1-1/d} \log n)}$*
4. *σ splits $O(\lambda^{1/d}n^{1-1/d})$ of the H_i, where λ is the ply of $\mathcal{H}$.*

Moreover, σ and $C'(\sigma, \mathcal{H})$ can be calculated in $2^{O(n^{1-1/d} \log n)}$ time.

Proof. Let σ be the separator given by Theorem 11. Then σ has properties 1 and 4. W.l.o.g., we assume that size$(\sigma) = 1$ and σ is centered at the origin. From now on, we will only consider edges that cross σ once and lie on the Hanan grid G. Any set $S \in C'(\sigma, \mathcal{H})$ we return can be divided into two subsets:

- $S_{short} := \{s \in S : length(s) \leq 1/n^{1/d}\}$
- $S_{long} := \{s \in S : length(s) > 1/n^{1/d}\}$

(The original proof uses three subsets: S_{short}, S_{mid} and S_{long}. However, since we have an extra factor $\log n$ in the exponent, we can merge S_{short} with part of S_{mid}, and merge the rest of S_{mid} with S_{long}.) We start with S_{short}. Let us take a look at a single facet of σ. We will now show that we can cross this facet in only a limited number of ways. First, we note that for the corresponding i, we have $|P_0^i(\sigma)| = O(n^{1-1/d})$. Let e be an arbitrary edge of our tour crossing σ through this face. Now, by property (3) of a canonical tour, the p_j and p_{j+1} that are connected by e must both have a distance at most $1/n^{1/d}$ to σ in the i'th coordinate. Therefore, the same holds of the odrdistances of the corresponding H_j. Furthermore, note that if we charge every edge e crossing σ through our facet to the two hypercubes of the corresponding p_j and p_{j+1}, no hypercube is charged more than twice. Hence, the number of short edges crossing σ through this facet is bounded by the number of hypercubes with odrdistance at most $1/n^{1/d}$ to σ, of which there are $O(n^{1-1/d})$. As there are $O(n^{d-1})$ possible locations for these edges to cross σ, there are $(n^{d-1})^{O(n^{1-1/d})} = 2^{O(n^{1-1/d} \log n)}$ possible combinations for every face. Finally, since there are $O(d)$ facets, the total amount of possible combinations is $2^{O(n^{1-1/d} \log n)}$.

We continue to S_{long}, the set of edges longer than $1/n^{1/d}$. Let us take a look at those edges which cross some arbitrary facet of σ. Using the same logic as in the original paper, by using Lemma 10, we can see that there are $(2n^{1/d})^{d-1} = O(n^{1-1/d})$ of these edges at most. Analogously, there are at most $O(n^{1-1/d})$ edges of which at least $1/(2n^{1/d})$ is inside σ. Since there are $O(n^{d-1})$ options for every edge, there are $(n^{d-1})^{O(n^{1-1/d})} = 2^{O(n^{1-1/d} \log n)}$ options for every face, and just as many for every σ. ◀

The algorithm. Our algorithm contains the following changes compared to the original:

- We choose our separator σ using Theorem 11 instead of the equivalent from the original paper, and use Theorem 12 to obtain the candidate sets of crossing points.
- Instead of choosing already existing points as boundary points, we create new boundary points as explained above. To ensure that the recursion ends, we bruteforce the solution if n is smaller than some arbitrarily large but fixed N, instead of recurring until $n = 1$.
- All H_i visited by one of the new boundary points are removed from both subproblems. For every H_i split by σ but not visited by one of the new boundary points, we need to "guess" whether it is visited inside or outside σ.
- For the initial call, we guess p_1 and p_n of the final tour, and connect them with a link - there are $O(n^{2d})$ viable combinations of points on the generalised Hanan grid. We remove all H_i we already visit by doing so, and then run the algorithm on the remaining H_i and the boundary set $B = \{p_1, p_n\}$.

Since there are $O(\lambda^{1/d})$ hypercubes H_i split by σ, there are $2^{O(\lambda^{1/d} n^{1-1/d} \log n)} \cdot 2^{O(\lambda^{1/d})} = 2^{O(\lambda^{1/d} n^{1-1/d} \log n)}$ subproblems generated in total, leading to the following theorem. (For the full proof, see Appendix D, where we show how to analyze the dependency on λ, and deal with the larger amount of candidate sets and the creation of extra boundary points.)

▶ **Theorem 13.** *Then Rectilinear One-of-a-Cube TSP on hypercubes with ply λ can be solved in $2^{O(\lambda^{\frac{1}{d}} n^{1-1/d} \log n)}$ time.*

References

1 Binay Bhattacharya, Ante Ćustić, Akbar Rafiey, Arash Rafiey, and Vladyslav Sokol. Approximation algorithms for generalized mst and TSP in grid clusters. In Zaixin Lu, Donghyun Kim, Weili Wu, Wei Li, and Ding-Zhu Du, editors, *Combinatorial Optimization and Applications*, pages 110–125, Cham, 2015. Springer International Publishing.

2 T.-H. Hubert Chan and Shaofeng H.-C. Jiang. Reducing curse of dimensionality: Improved PTAS for TSP (with neighborhoods) in doubling metrics. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 754–765. SIAM, 2016. doi:10.1137/1.9781611974331.ch54.

3 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, and Sudeshna Kolay. An ETH-tight exact algorithm for euclidean TSP. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 450–461, 2018. doi:10.1109/FOCS.2018.00050.

4 Mark de Berg, Joachim Gudmundsson, Matthew J. Katz, Christos Levcopoulos, Mark H. Overmars, and A. Frank van der Stappen. TSP with neighborhoods of varying size. *Journal of Algorithms*, 57(1):22–36, 2005. doi:10.1016/j.jalgor.2005.01.010.

5 Moshe Dror and James B. Orlin. Combinatorial optimization with explicit delineation of the ground set by a collection of subsets. *SIAM Journal on Discrete Mathematics*, 21(4):1019–1034, 2008. doi:10.1137/050636589.

6 M. R. Garey, Ronald L. Graham, and David S. Johnson. Some NP-complete geometric problems. In *STOC*, pages 10–22. ACM, 1976.

7 Gregory Gutin and Abraham P. Punnen. *The Traveling Salesman Problem and Its Variations*. Springer, 2006.

8 R. Z. Hwang, R. C. Chang, and Richard C. T. Lee. The searching over separators strategy to solve some NP-hard problems in subexponential time. *Algorithmica*, 9(4):398–423, 1993. doi:10.1007/BF01228511.

9 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.

10 Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology Stockholm, 1992.

11 Michael Khachay and Katherine Neznakhina. Approximation algorithms for generalized TSP in grid clusters. *CEUR Workshop Proceedings*, 1623:39–48, 2016.

12 Michael Khachay and Katherine Neznakhina. Complexity and approximability of the euclidean generalized traveling salesman problem in grid clusters. *Annals of Mathematics and Artificial Intelligence*, 88(1):53–69, 2020. doi:10.1007/s10472-019-09626-w.

13 Joseph S. B. Mitchell. A PTAS for TSP with neighborhoods among fat regions in the plane. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007*, pages 11–18, 2007.

14 Joseph S.B. Mitchell. A constant-factor approximation algorithm for TSP with pairwise-disjoint connected neighborhoods in the plane. In *Proceedings of the Twenty-Sixth Annual Symposium on Computational Geometry*, pages 183–191, 2010. doi:10.1145/1810959.1810992.

15 Christos H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theor. Comput. Sci.*, 4(3):237–244, 1977.

16 Warren D. Smith and Nicholas C. Wormald. Geometric separator theorems & applications. In *FOCS*, pages 232–243. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743449.

A Running time of the One-of-a-Set TSP algorithm

To prove the running time of the One-of-a-Set TSP algorithm, we can follow the proof of the running time of the the original algorithm [3] almost verbatim. We only need to take the dependency on λ into account at the right places. Define $T(n, b)$ to be the running time of the algorithm when run on an input containing n points of which b are boundary points. Let $n_{S,in}$, $n_{S,out}$, $b_{S,in}$ and $b_{S,out}$ denote the numbers of points and boundary points in the subproblems generated.

As far as the candidate sets are concerned, we can restrict our attention to candidate sets $S \in \mathcal{C}'(\sigma, P)$ that contain at most one edge incident to any given point in B, and at most two edges incident to any given point in $P \setminus B$. Define $\mathcal{C}''(\sigma, \mathcal{P})$ to be the family of candidate sets gained from restricting $\mathcal{C}'(\sigma, P)$ this way. Furthermore, given a candidate set and a boundary set, for every point set split by σ we need to choose whether to use a point in σ_{in} or σ_{out}. Let $I(\sigma, \mathcal{P})$ be the set of 2^m possible combinations of choices, where m is the number of point sets split by σ. Clearly, if one of the points of a P_i is in B, no choice needs to be made. Define $\mathcal{C}^*(\sigma, \mathcal{P}) \subseteq \mathcal{C}''(\sigma, P) \times I(\sigma, \mathcal{P})$ to be the set of combinations of candidate sets and splitting choices restricted this way.

We get:

$$T(n,b) \leq \begin{cases} c_0 & \text{if } n \leq 1 \\ \sum_{S \in \mathcal{C}^*(\sigma_{\mathcal{P}}, \mathcal{P})} e^{c_3(n^{1-1/d}+b)} + T(n_{S,in}, b_{S,in}) + T(n_{S,out}, b_{S,out}) & \text{if } b \leq \gamma n^{1-1/d} \\ \sum_{S \in \mathcal{C}^*(\sigma_B, \mathcal{P})} e^{c_3(n^{1-1/d}+b)} + T(n_{S,in}, b_{S,in}) + T(n_{S,out}, b_{S,out}) & \text{if } b > \gamma n^{1-1/d}. \end{cases}$$

We prove by induction that $T(n,b) \leq e^{\lambda^{\frac{1}{d}}(d_1 n^{1-1/d}+d_2 b)}$ for some constants d_1 and d_2 and for all $1 \leq b \leq n$. This clearly holds for $b, n \leq 1$, so by induction, for each S we have

$$\begin{aligned} & e^{c_3(n^{1-1/d}+b)} + T(n_{S,in}, b_{S,in}) + T(n_{S,out}, b_{S,out}) \\ \leq\ & e^{c_3(n^{1-1/d}+b)} + e^{\lambda^{\frac{1}{d}}(d_1 n_{S,in}^{1-1/d}+d_2 b_{S,in})} + e^{\lambda^{\frac{1}{d}}(d_1 n_{S,out}^{1-1/d}+d_2 b_{S,out})}. \end{aligned}$$

Let c_2, c_4 and c_5 be the constants from part (2), (3) and (4) of Theorem 2. For any $S \in \mathcal{C}^*(\sigma_B, \mathcal{P})$, we have $b_{S,in} \leq \delta b + c_2 n^{1-1/d}$ and $b_{S,out} \leq \delta b + c_2 n^{1-1/d}$; similarly, for any $S \in \mathcal{C}^*(\sigma_{\mathcal{P}}, \mathcal{P})$, we have $n_{S,in} \leq \delta n$ and $n_{S,out} \leq \delta n$. In the remaining cases, we can just use the trivial bounds $b_{S,\cdot} \leq b + c_2 n^{1-1/d}$ and $n_{S,\cdot} \leq n$. Since $|\mathcal{C}^*(\sigma, \mathcal{P})| \leq e^{c_4 \lambda^{\frac{1}{d}} n^{1-1/d}}$, we get the following:

$$T(n,b) \leq \begin{cases} c_0 & \text{if } n \leq 1 \\ \sum_{S \in \mathcal{C}^*(\sigma_{\mathcal{P}}, \mathcal{P})} e^{c_3(n^{1-1/d}+b)} + T(n_{S,in}, b_{S,in}) + T(n_{S,out}, b_{S,out}) & \text{if } b \leq \gamma n^{1-1/d} \\ \sum_{S \in \mathcal{C}^*(\sigma_B, \mathcal{P})} e^{c_3(n^{1-1/d}+b)} + T(n_{S,in}, b_{S,in}) + T(n_{S,out}, b_{S,out}) & \text{if } b > \gamma n^{1-1/d}. \end{cases}$$

$$T(n,b) \leq \begin{cases} c_0 & \text{if } n \leq 1 \\ e^{c_4 \lambda^{\frac{1}{d}} n^{1-1/d}} \left(e^{c_3(n^{1-1/d}+b)} + 2e^{\lambda^{\frac{1}{d}}(d_1(\delta n)^{1-1/d} + d_2(b + c_2 n^{1-1/d}))} \right) & \text{if } b \leq \gamma n^{1-1/d} \\ e^{c_4 \lambda^{\frac{1}{d}} n^{1-1/d}} \left(e^{c_3(n^{1-1/d}+b)} + 2e^{\lambda^{\frac{1}{d}}(d_1 n^{1-1/d} + d_2(\delta b + c_2 n^{1-1/d}))} \right) & \text{if } b > \gamma n^{1-1/d}. \end{cases}$$

For simplicity, let $\lambda' := \lambda^{\frac{1}{d}}$, and Let $c := \max\{c_0, c_2, c_3, c_4\}$. We get

$$T(n,b) \leq \begin{cases} c & \text{if } n \leq 1 \\ e^{c\lambda' n^{1-1/d} + c(n^{1-1/d}+b) + 1 + \lambda'(d_1(\delta n)^{1-1/d} + d_2(b + cn^{1-1/d}))} & \text{if } b \leq \gamma n^{1-1/d} \\ e^{c\lambda' n^{1-1/d} + c(n^{1-1/d}+b) + 1 + \lambda'(d_1 n^{1-1/d} + d_2(\delta b + cn^{1-1/d}))} & \text{if } b > \gamma n^{1-1/d}. \end{cases}$$

$$T(n,b) \leq \begin{cases} c & \text{if } n \leq 1 \\ e^{\lambda'\left((2c + cd_2 + d_1\delta^{1-1/d}) n^{1-1/d} + (d_2 + 2c)b\right)} & \text{if } b \leq \gamma n^{1-1/d} \\ e^{\lambda'\left((2c + cd_2 + d_1) n^{1-1/d} + (d_2\delta + 2c)b\right)} & \text{if } b > \gamma n^{1-1/d}. \end{cases}$$

Now, for the case $b > \gamma n^{1-1/d}$, we have

$$e^{\lambda'\left((2c + cd_2 + d_1) n^{1-1/d} + (d_2\delta + 2c)b\right)} \leq e^{\lambda'\left(d_1 n^{1-1/d} + d_2 b\right)}$$

if and only if

$$(2c + cd_2)n^{1-1/d} + (d_2(\delta - 1) + 2c)b \leq (2c + d_2(c - \gamma(1-\delta)) + 2c\gamma)n^{1-1/d} \leq 0.$$

We choose $\gamma = \frac{2c}{1-\delta}$ and $d_2 = 2 + 2\gamma$, satisfying the equation.

For the case $b \leq \gamma n^{1-1/d}$, we have

$$e^{\lambda'\left((2c+cd_2+d_1\delta^{1-1/d})n^{1-1/d}+(d_2+2c)b\right)} \leq e^{\lambda'(d_1 n^{1-1/d}+d_2 b)}$$

if and only if

$$(2c + cd_2 + d_1(\delta^{1-1/d} - 1))n^{1-1/d} + 2cb \leq (2c + cd_2 + d_1(\delta^{1-1/d} - 1) + 2c\gamma)n^{1-1/d} \leq 0.$$

We choose $d_1 = \frac{4c(1+\gamma)}{1-\delta^{1-1/d}}$, satisfying the equation.

Finally, we note that d_1 and d_2 are indeed (nonnegative) constants, as they only depend on c, γ and δ, which in turn only depend on d.

B Proof of Lemma 8

Let T be any shortest tour with a minimal number of edges on the given hypercubes $H_1, ..., H_n$. Note that because T has a minimal number of edges, no edges have length 0. Recall that p^i denotes the i'th coordinate of p. For simplicity, let us call the first coordinate the *x-coordinate*, and let us call those edges whose endpoints have different x-coordinates *horizontal*. For every H_i, let x_{i1} and x_{i2} denote the x-coordinates of the corner points of H_i. We will now show that we can change T into a shortest tour of which all x-coordinates of the endpoints of the edges used are in the set $X_H := \{x_{ij} | i \in \{1, ..., n\}, j \in \{1, 2\}\}$. Furthermore, we do so without changing the sets used for the second to d'th coordinate. Then, by applying this method repeatedly, we obtain a shortest tour of which all coordinates match those of the corner points of the H_i, i.e., a shortest tour which lies on the generalised Hanan grid.

Let $X_T := \{x_1 < ... < x_r\}$ be the set of x-coordinates used by T. For every x_i, let E_i be the set of horizontal edges of with an endpoint with x-coordinate x_i.

Let x_i be an x-coordinate not in X_H. Then let e_1 and e_2 be two consecutive edges in E_1 (consecutive as in there are no edges in E_1 in between e_1 and e_2 in T). Let E denote the set of edges between e_1 and e_2 in T. Note that all endpoints of these edges have x-coordinate x_i. W.l.o.g., let at least one of endpoints of e_1 and e_2 lie to the right of x_i. Let x' denote the smallest x-coordinate in $X_H \cup X_T$ strictly larger than x_i.

Now, let us change the x-coordinate of all edges in E to x'. Furthermore, we move the endpoints of e_1 and e_2 with x-coordinate x_i to the x-coordinate x'. See Figure 7 for an example. Note that the resulting tour T' is indeed still a tour. Furthermore, T' visits all H_i: Let p be an arbitrary point in H_i visited by T but not by T'. Note that this is only possible if $x_i \leq p^0 x'$. However, in that case, the point $p' = (x', p^1, ..., p^n)$ is in H_i as well, and p' is visited by T'.

Now, if both other endpoints of e_1 and e_2 are on the same side of the hyperplane defined by x-coordinate x_i, the resulting tour T' is strictly shorter than T. Since T is a shortest tour, we conclude that the other endpoints of e_1 and e_2 are on different sides of the hyperplane defined by x-coordinate x_i. Furthermore, note that the edge that has been shortened still has a positive length, otherwise the assumption that T has a minimal number of edges fails. Finally, note that this change does not change the set of all other coordinates used except the x-coordinates.

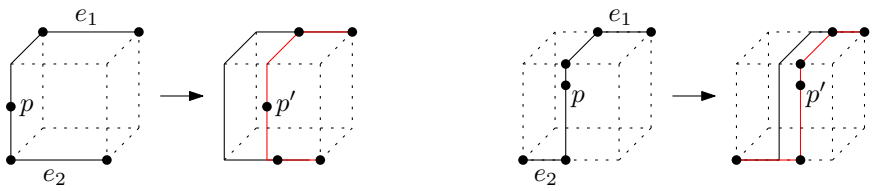


Figure 7 An example for the proof of Lemma 8. In the left case, we can shorten the tour T by shortening the edges e_1 and e_2 and moving the connected edges correspondingly. As long as the x-coordinate of these points was not a coordinate in X_H, if a point p is in a hypercube, then so is p'. In the right case, we are free to move the edges between e_1 and e_2 to the smallest x-coordinate bigger than x_i, as long as their x-coordinate is not a coordinate in X_H.

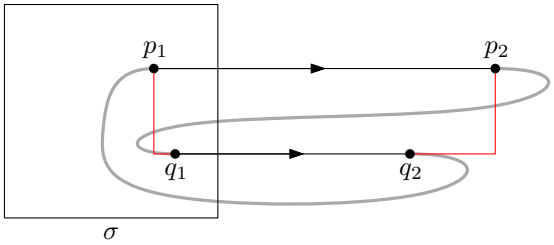


Figure 8 An example for the proof of Lemma 10. If there are enough (directed) edges of length at least $size(\sigma)$ crossing σ, there must be two edges (p_1, p_2) and (q_1, q_2) going in the same direction, crossing the same face, both with at least a length of $size(\sigma)/2$ on the same side of this face, and with $|p_1q_1| < size(\sigma)/2$. We can then create a strictly shorter tour by removing both edges and connecting p_1 to q_1 and q_2 to p_2 (in red). The resulting set of edges is indeed a tour, if we flip the direction of the edges between p_2 and q_1.

Now, we can apply the above change exhaustively: in every step, we increase the sum of all x-coordinates of all endpoints of all edges in T by at least some amount dependent only on $X_H \cup X_T$, and the total sum is bounded by $2n$ times the maximum x-coordinate in $X_H \cup X_T$. After applying this change exhaustively, no more x-coordinates not in X_H are used.

Since this procedure does not change the set of other coordinates used, we can apply this procedure once for every coordinate, obtaining a T' which lies on the generalised Hanan grid.

C Proof of Lemma 10

Let T be a shortest (directed) rectilinear tour on the hypercubes $H_1, ..., H_n$. Let σ be a separator. We will first show that T contains $O(1)$ edges crossing σ of length at least $size(\sigma)$. W.l.o.g., assume $size(\sigma) = 1$. Now, suppose T contains at least $2d \cdot 4 \cdot 4^d$ edges crossing σ of length at least $size(\sigma)$. Then there exists a face f of σ such that at least $4 \cdot 4^d$ edges cross f. W.l.o.g., at least $2 \cdot 4^d$ edges cross f from σ_{in} to σ_{out}. W.l.o.g., at least 4^d of these edges have length at least $1/2$ outside σ. Therefore, there must be two of these edges (p_1, p_2) and (q_1, q_2) with $|p_1, q_1| < 1/2$. Recall that $|pq|$ denotes the rectilinear distance between p and q. See Figure 8 for an example. Let us remove these two edges, and connect p_1 to q_1 and q_2 to q_2. Next, we flip the direction of the edges from p_2 to q_1. We claim that the resulting tour T' is a strictly shorter rectilinear tour visiting all H_i. Since this directly contradicts our assumption, we can then conclude that T contains $O(1)$ edges crossing σ of length at least $size(\sigma)$.

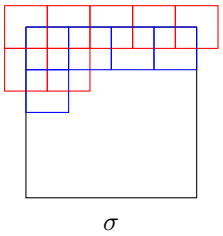


Figure 9 An example for the proof of Lemma 10. In black, the separator σ. In red and blue, some of the smaller hypercubes covering σ_{in}. Any edge of length at least $size(\sigma)/4$ crosses at least one of the smaller hypercubes. Since there are $O(1)$ smaller hypercubes, each being crossed $O(1)$ time,s there are $O(1)$ edges of length at least $size(\sigma)/4$ of T in σ_{in}.

First, we note that T' is indeed a tour: see Figure 8 for an example. Furthermore, T' indeed visits all H_i: since T is a simple tour, any tour visiting all endpoints of the edges of T (and hence, the points $p_1, ..., p_n$) visits all H_i. Finally, T' is strictly shorter than T: clearly,

$$||T|| - ||T'|| = |p_1p_2| + |q_1q_2| - |p_1q_1| - |p_2q_2|.$$

W.l.o.g., let $|p_1p_2| \geq |q_1q_2|$. We know that $|p_1p_2 \geq |q_1q_2| \geq 1$. Furthermore, we know that $|p_1q_1| < 1/2$. Since p_1p_2 and q_1q_2 both go in the same direction, we get

$$|p_2q_2| \leq |p_1q_1| + |p_1p_2| - |q_1q_2|.$$

Combining these, we get

$$\begin{aligned} ||T|| - ||T'|| &= |p_1p_2| + |q_1q_2| - |p_1q_1| - |p_2q_2| \\ &\geq |p_1p_2| + |q_1q_2| - |p_1q_1| - (|p_1q_1| + |p_1p_2| - |q_1q_2|) \\ &\geq 2|q_1q_2| - 2|p_1q_1| \\ &\geq 2 \cdot 1 - 2 \cdot 1/2 > 0, \end{aligned}$$

as we wanted to prove.

Next, we show that T contains $O(1)$ edges fully in σ_{in} with length at least $size(\sigma)/4$. W.l.o.g., let σ be the hypercube of size 1 with center $c = (1/2, ..., 1/2)$. For $0 \leq i_1, ..., i_d \leq 8$, let $\sigma_{i_1,...,i_d}$ be the hypercube of size $1/4$ with center $(i_1/8, ..., i_2/8)$. Then, every edge of T fully in σ_{in} of length at least $1/4$ crosses at least one of these hypercubes; see Figure 9 for an example. On the other hand, by using the first part of this proof we conclude that every one of these smaller hypercubes is crossed $O(1)$ times. Since the number of smaller hypercubes is $9^d = O(1)$, we conclude that there are $O(1)$ edges of T of length at least $size(\sigma)/4$ fully in σ_{in}. This concludes the proof of the second part of the Packing Property, and hence, the proof of the Packing Property for edges of a simple tour.

D Running time of the Rectilinear One-of-a-Cube TSP algorithm

There are three differences between the algorithms that impact the running time. First, as mentioned, there are n^{2d} initial calls made to the algorithm, one for every pair of points. However, since we will prove that the running time is $2^{O(\lambda^{1/d}n^{1-1/d}\log n)}$, this factor is irrelevant. Second is the fact that there are more candidate sets. Specifically, $2^{O(\lambda^{1/d}n^{1-1/d}\log n)}$ subproblems are generated. Finally, because we only guess where the separator is crossed, $O(n^{1-1/d})$ new boundary points are generated instead of selected from the already existing points. We will now compute the impact of the last two differences on the running time of the algorithm.

Define $T(n, b)$ to be the running time of the algorithm when run on an input containing n points of which b are boundary points. Let $n_{S,in}$, $n_{S,out}$, $b_{S,in}$ and $b_{S,out}$ denote the numbers of points and boundary points in the subproblems generated.

Let $I(\sigma, \mathcal{H})$ be the set of 2^m possible combinations of choices, where m is the number of hypercubes split by σ. Clearly, if one of the points of an H_i is in B, no choice needs to be made. Define $\mathcal{C}^*(\sigma, \mathcal{H}) \subseteq \mathcal{C}'(\sigma, H) \times I(\sigma, \mathcal{H})$ to be the set of combinations of candidate sets and splitting choices restricted this way.

Let N be arbitrarily large but fixed. We get:

$$T(n,b) \leq \begin{cases} c_0 & \text{if } n \leq N \\ \sum_{S \in \mathcal{C}^*(\sigma_{\mathcal{H}}, \mathcal{H})} e^{c_3(n^{1-1/d}+b)\log n} + T(n_{S,in}, b_{S,in}) + T(n_{S,out}, b_{S,out}) & \text{if } b \leq \gamma n^{1-1/d} \\ \sum_{S \in \mathcal{C}^*(\sigma_B, \mathcal{H})} e^{c_3(n^{1-1/d}+b)\log n} + T(n_{S,in}, b_{S,in}) + T(n_{S,out}, b_{S,out}) & \text{if } b > \gamma n^{1-1/d}. \end{cases}$$

We prove by induction that $T(n, b) \leq e^{\lambda^{\frac{1}{d}}(d_1 n^{1-1/d} + d_2 b)\log n}$ for some constants d_1 and d_2 and for all $1 \leq b \leq n$. This clearly holds for $b, n \leq N$, so by induction, for each S we have

$$\begin{aligned} & e^{c_3(n^{1-1/d}+b)\log n} + T(n_{S,in}, b_{S,in}) + T(n_{S,out}, b_{S,out}) \\ \leq\ & e^{c_3(n^{1-1/d}+b)\log n} + e^{\lambda^{\frac{1}{d}}(d_1 n_{S,in}^{1-1/d} + d_2 b_{S,in})\log n} + e^{\lambda^{\frac{1}{d}}(d_1 n_{S,out}^{1-1/d} + d_2 b_{S,out})\log n}. \end{aligned}$$

Let c_2, c_4 and c_5 be the constants from part (2), (3) and (4) of Theorem 12. For any $S \in \mathcal{C}^*(\sigma_B, \mathcal{H})$, we have $b_{S,in} \leq \delta b + c_2 n^{1-1/d}$ and $b_{S,out} \leq \delta b + c_2 n^{1-1/d}$; similarly, for any $S \in \mathcal{C}^*(\sigma_{\mathcal{H}}, \mathcal{H})$, we have $n_{S,in} \leq \delta n + c_2 n^{1-1/d}$ and $n_{S,out} \leq \delta n + c_2 n^{1-1/d}$. In the remaining cases, for $b_{S,\cdot}$ we can use the trivial bound $b_{S,\cdot} \leq b + c_2 n^{1-1/d}$. For $n_{S,\cdot}$, we can use $n_{S,\cdot} \leq n$; despite the possibility of new points being created, there will never be more points created then there are points on either side of σ. Since $|\mathcal{C}^*(\sigma, \mathcal{H})| \leq e^{c_4 \lambda^{\frac{1}{d}} n^{1-1/d} \log n}$, we get the following:

$$T(n,b) \leq \begin{cases} c_0 & \text{if } n \leq N \\ \sum_{S \in \mathcal{C}^*(\sigma_{\mathcal{H}}, \mathcal{H})} e^{c_3(n^{1-1/d}+b)\log n} + T(n_{S,in}, b_{S,in}) + T(n_{S,out}, b_{S,out}) & \text{if } b \leq \gamma n^{1-1/d} \\ \sum_{S \in \mathcal{C}^*(\sigma_B, \mathcal{H})} e^{c_3(n^{1-1/d}+b)\log n} + T(n_{S,in}, b_{S,in}) + T(n_{S,out}, b_{S,out}) & \text{if } b > \gamma n^{1-1/d}. \end{cases}$$

$$\leq \begin{cases} c_0 & \text{if } n \leq N \\ e^{c_4 \lambda^{\frac{1}{d}} n^{1-1/d} \log n} \left(e^{c_3(n^{1-1/d}+b)\log n} + 2e^{\lambda^{\frac{1}{d}}(d_1(\delta n + c_2 n^{1-1/d})^{1-1/d} + d_2(b + c_2 n^{1-1/d}))\log n} \right) & \text{if } b \leq \gamma n^{1-1/d} \\ e^{c_4 \lambda^{\frac{1}{d}} n^{1-1/d} \log n} \left(e^{c_3(n^{1-1/d}+b)\log n} + 2e^{\lambda^{\frac{1}{d}}(d_1 n^{1-1/d} + d_2(\delta b + c_2 n^{1-1/d}))\log n} \right) & \text{if } b > \gamma n^{1-1/d}. \end{cases}$$

For simplicity, let $\lambda' := \lambda^{\frac{1}{d}}$, let $n' := n^{1-1/d}$, and Let $c := \max\{c_0, c_2, c_3, c_4\}$. We get

$$T(n,b) \leq \begin{cases} c & \text{if } n \leq N \\ e^{c\lambda' n' \log n + c(n'+b)\log n + 1 + \lambda'(d_1(\delta n + cn')^{1-1/d} + d_2(b+cn'))\log n} & \text{if } b \leq \gamma n' \\ e^{c\lambda' n' \log n + c(n'+b)\log n + 1 + \lambda'(d_1 n' + d_2(\delta b + cn'))\log n} & \text{if } b > \gamma n'. \end{cases}$$

Now, if n is large enough (dependent only on d), then $\delta n + cn' \leq \zeta n$, where $\zeta = \frac{1+\delta}{2}$. Since we know that $n > N$ and N is arbitrarily large, we get

$$T(n,b) \leq \begin{cases} c & \text{if } n \leq N \\ e^{c\lambda' n' \log n + c(n'+b)\log n + 1 + \lambda'(d_1 \zeta^{1-1/d} n' + d_2(b+cn'))\log n} & \text{if } b \leq \gamma n' \\ e^{c\lambda' n' \log n + c(n'+b)\log n + 1 + \lambda'(d_1 n' + d_2(\delta b + cn'))\log n} & \text{if } b > \gamma n'. \end{cases}$$

$$T(n,b) \leq \begin{cases} c & \text{if } n \leq N \\ e^{(c\lambda'+c+\lambda' d_1 \zeta^{1-1/d}+\lambda' d_2 c)n' \log n+(cb+\lambda' d_2 b)\log n+1} & \text{if } b \leq \gamma n' \\ e^{(c\lambda'+c+\lambda' d_1+\lambda' d_2 c)n' \log n+(cb+\lambda' d_2 \delta b)\log n+1} & \text{if } b > \gamma n'. \end{cases}$$

$$T(n,b) \leq \begin{cases} c & \text{if } n \leq N \\ e^{\lambda' \log n((\zeta^{1-1/d} d_1+3cd_2)n'+(2c+d_2)b)} & \text{if } b \leq \gamma n' \\ e^{\lambda' \log n((d_1+3cd_2)n'+(2c+\delta d_2)b)} & \text{if } b > \gamma n'. \end{cases}$$

Now, for the case $b > \gamma n'$, we have

$$e^{\lambda' \log n((d_1+3cd_2)n'+(2c+\delta d_2)b)} \leq e^{\lambda'\left(d_1 n'+d_2 b\right)\log n}$$

if and only if

$$(d_1 + 3cd_2)n' + (2c + \delta d_2)b \leq d_1 n' + d_2 b.$$

We choose $\gamma = \frac{4c}{1-\delta}$ and $d_2 = \frac{8c}{1-\delta}$, satisfying the equation.

For the case $b \leq \gamma n^{1-1/d}$, we have

$$e^{\lambda' \log n((\zeta^{1-1/d} d_1+3cd_2)n'+(2c+d_2)b)} \leq e^{\lambda'(d_1 n'+d_2 b)\log n}$$

if and only if

$$(\zeta^{1-1/d} d_1 + 3cd_2)n' + (2c + d_2)b \leq d_1 n' + d_2 b.$$

We choose $d_1 = \frac{32c^2}{(1-\zeta^{1-1/d})(1-\delta)}$, satisfying the equation.

Finally, we note that γ, d_1 and d_2 are indeed (nonnegative) constants, as they only depend on c, δ and ζ, which in turn only depend on d.

Depth-Three Circuits for Inner Product and Majority Functions

Kazuyuki Amano
Gunma University, Kiryu, Japan

Abstract

We consider the complexity of depth-three Boolean circuits with limited bottom fan-in that compute some explicit functions. This is one of the simplest circuit classes for which we cannot derive tight bounds on the complexity for many functions. A Σ_3^k-circuit is a depth-three OR $\circ$ AND $\circ$ OR circuit in which each bottom gate has fan-in at most k.

First, we investigate the complexity of Σ_3^k-circuits computing the inner product mod two function IP_n on n pairs of variables for small values of k. We give an explicit construction of a Σ_3^2-circuit of size smaller than $2^{0.952n}$ for IP_n as well as a Σ_3^3-circuit of size smaller than $2^{0.692n}$. These improve the known upper bounds of $2^{n-o(n)}$ for Σ_3^2-circuits and $3^{n/2} \sim 2^{0.792n}$ for Σ_3^3-circuits by Golovnev, Kulikov and Williams (ITCS 2021), and also the upper bound of $2^{(0.965\ldots)n}$ for Σ_3^2-circuits shown in a recent concurrent work by Göös, Guan and Mosnoi (MFCS 2023).

Second, we investigate the complexity of the majority function MAJ_n aiming for exploring the effect of negations. Currently, the smallest known depth-three circuit for MAJ_n is a monotone circuit. A $\Sigma_3^{(+k,-\ell)}$-circuit is a Σ_3-circuit in which each bottom gate has at most k positive literals and ℓ negative literals as its input. We show that, for $k \leq 2$, the minimum size of a $\Sigma_3^{(+k,-\infty)}$-circuit for MAJ_n is essentially equal to the minimum size of a monotone Σ_3^k-circuit for MAJ_n. In sharp contrast, we also show that, for $k = 3, 4$ and 5, there exists a $\Sigma_3^{(+k,-\ell)}$-circuit computing MAJ_n (for an appropriately chosen ℓ) that is smaller than the smallest known monotone Σ_3^k-circuit for MAJ_n. Our results suggest that negations may help to speed up the computation of the majority function even for depth-three circuits. All these constructions rely on efficient circuits or formulas on a small number of variables that we found through a computer search.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases Circuit complexity, depth-3 circuits, upper bounds, lower bounds, computer-assisted proof

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.7

Supplementary Material *Dataset*: `https://gitlab.com/KazAmano/depth-3-circuits`
archived at `swh:1:dir:6c19f0a1a69eae1143d2270517dd6f46df08bfb7`

Funding This work was supported in part by JSPS Kakenhi No. JP21K19758, JP18K11152 and JP18H04090.

Acknowledgements The author would like to thank anonymous referees for their helpful comments.

1 Introduction

Deriving a strong lower bound on the size of a Boolean circuit computing an explicit function is one of the most challenging problems in theoretical computer science. Many different types of restricted circuits have been investigated, and this paper concentrates on depth-three circuits.

A Σ_3-circuit is a depth-three OR $\circ$ AND $\circ$ OR circuit consisting of unbounded fan-in AND/OR gates, with variables or their negations feeding into the bottom gates. In other words, a Σ_3-circuit is an OR of an arbitrary number of CNF formulas.

Despite its simplicity, there still is a considerable gap between the upper and lower bounds on the size of Σ_3-circuits. Here, the *size* of a Σ_3-circuit is defined as the number of *gates* in the circuit. A counting argument shows that a random function on n variable needs a Σ_3-circuit of size $\Theta(2^{n/2})$ [6, 25]. However, the strongest lower bound on the size of a Σ_3-circuit for an explicit Boolean function is $2^{\Omega(\sqrt{n})}$ (e.g., [16] for the proof for the parity and majority functions). Deriving a lower bound of $2^{\omega(\sqrt{n})}$ on the size of a Σ_3-circuit computing some explicit function has been open for over 30 years.

Such a situation motivates us to consider further restricted Σ_3-circuits. One natural restriction is to bound the bottom fan-in. For a natural number k, a Σ_3^k-circuit is a Σ_3-circuit with bottom fan-in bounded by k, or equivalently, an OR of k-CNF formulas.

When the value of k is small, stronger lower bounds are known. For example, Paturi, Saks and Zane [21] showed that the minimum size of a Σ_3^k-circuit computing the parity function on n variables is at least $2^{n/k}$, for any $k \leq O(\sqrt{n})$. See e.g., the introduction of [17] or [11] for more results on Σ_3^k-circuits. A recent work by Golovnev, Kulikov and Williams [11] showed that a $2^{n-o(n)}$ lower bound on the Σ_3^{16}-circuit size for an explicit function implies a $3.9n$ lower bound on the general circuit size, which would be a breakthrough on circuit complexity since the best known lower bound is much smaller, say, $3.1n - o(n)$ [20] (see also [8]).

Despite the simplicity of a model, for many functions, we still do not know the size of an optimal Σ_3^k-circuit even for small values of k. In this paper, we investigate Σ_3^k-circuits for two well-studied functions, namely, the inner product mod two function and the majority function.

1.1 Inner Product

The first target function we consider in this paper is the *inner product mod two* function $\mathsf{IP}_n(x_1, \ldots, x_n, y_1, \ldots, y_n) := \oplus_i x_i y_i$. The function IP_n has frequently appeared as a target for analyzing the complexity of shallow circuits (see e.g., [2, 9, 11, 15, 18]).

For a Boolean function f, we write the minimum size (i.e., number of gates) of a Σ_3^k-circuit computing f as $s_3^k(f)$. For many functions f, we do not have a technique for determining $s_3^k(f)$ even for small values of k. We sometimes consider the minimum fan-in of the top OR gate in a Σ_3^k-circuit computing f, which is denoted by $\tilde{s}_3^k(f)$. It is easy to see that $s_3^k(f)$ is at most polynomially larger than $\tilde{s}_3^k(f)$, when $k = O(1)$.

Recently, Golovnev, Kulikov and Williams [11] and Frankl, Gryaznov and Talebanfard [10] invesigated the complexity of IP_n for Σ_3^2 and Σ_3^3-circuits. The bounds described in [10, 11] are $2^{n/2} \leq s_3^2(\mathsf{IP}_n) \leq 2^{n-o(n)}$ and $2^{n/3} \leq s_3^3(\mathsf{IP}_n) \leq 3^{n/2} \sim 2^{0.792n}$. Both upper bounds are given in [11]. Both lower bounds are via a simple reduction to the parity function $\oplus_n$ on n variables and the fact that $s_3^k(\oplus_n) \geq 2^{n/k}$ [21]. The problem of determining $s_3^3(\mathsf{IP}_n)$ as well as $s_3^2(\mathsf{IP}_n)$ has been left as an open problem in these works. After the initial submission of this manuscript, we learned that Göös, Guan and Mosnoi [12] subsequently improved the upper and lower bounds on the size of Σ_3^2-circuits to $2^{(0.847\ldots)n} < s_3^2(\mathsf{IP}_n) < 2^{(0.965\ldots)n}$ using an LP-based technique.

In this work, we show that both upper bounds can be improved considerably. Namely, we present an explicit construction of Σ_3^2-circuits of size less than $2^{0.952n}$ and Σ_3^3-circuits of size less than $2^{0.692n}$ that compute IP_n (we will review this more carefully in Section 1.3).

1.2 Majority

The second target function we consider in this paper the size of is the *majority* function $\mathsf{MAJ}_n(x_1, \ldots, x_n) := [\sum_i x_i \geq n/2]$, where $[\cdot]$ denotes the Iverson bracket.

The best known upper bound on the size of a Σ_3-circuit for MAJ_n is $2^{O(\sqrt{n \log n})}$ [3, 19]. Note that their circuit is monotone, i.e., a circuit without negative literals. The best known lower bound on the size of a Σ_3-circuit for MAJ_n is $2^{d(\sqrt{n})-o(\sqrt{n})}$ where $d = 1/\sqrt{\ln 4} = 0.849\ldots$ due to Håstad, Jukna and Pudlák [16]. Hence, there still is $\sqrt{\log n}$ factor gap in the exponent and there is a possibility that the minimum size of a Σ_3-circuit for MAJ_n is $2^{\omega(\sqrt{n})}$. A recent work by Cavalar and Oliveira [4] reveals that a slightly weaker lower bound of $2^{\Omega(\sqrt{n/\log n})}$ can be shown via monotone simulations of non-monotone circuits.

Apparently, one natural question is whether an optimal Σ_3-circuit (or Σ_3^k-circuit) for MAJ_n is monotone or not, which is the main focus of the second part of this work.

The resolution of this problem would contribute to unraveling the mystery of NOT gates in the computation of Boolean functions. Despite more than 30 years have passed since exponential lower bounds on the size of monotone circuits were proven [1, 23], we cannot prove even $4n$ lower bound on a general circuit size for an explicit function in NP.

The depth-three is the simplest interesting case in the sense that negations are known to be useless for computing monotone functions in depth-two circuits. Namely, Quine [22] showed that, for any monotone Boolean function f, a smallest DNF (or CNF, respectively) computing f is a monotone DNF (or monotone CNF, respectively).

For two non-negative integers k and ℓ, a $(+k,-\ell)$-*CNF formula* is a CNF formula in which each clause contains at most k positive literals and at most ℓ negative literals. A $\Sigma^{(+k,-\ell)}$-circuit is a Σ_3-circuit in which all inputs of the top OR gate are $(+k,-\ell)$-CNF formulas. A $\Sigma_3^{(+k,-0)}$-circuit, which is a monotone Σ_3^k-circuit, is simply written as a Σ_3^{+k}-circuit. It is tempting to guess that, for any ℓ, an optimal $\Sigma_3^{(+k,-\ell)}$-circuit for MAJ_n is in fact a monotone, i.e., a Σ_3^{+k}-circuit.

In this work, we obtain some evidence suggesting that this hypothesis may not hold for $k \geq 3$. A detailed explanation of what we have shown is provided in the next subsection.

1.3 Our Contributions and Implications

In short, the contribution of this paper is to improve the upper bounds on the size of a Σ_3-circuit with limited bottom fan-in that compute IP_n and MAJ_n.

For IP_n, we give an explicit construction showing that (i) $\tilde{s}_3^2(\mathsf{IP}_n) < 2^{0.952n}$, and (ii) $\tilde{s}_3^3(\mathsf{IP}_n) < 2^{0.692n}$. As described in Section 1.1, these improve the known upper bounds in [11] and [12] . Note that the construction of our circuits includes components that are found by a computer search. As a result, some circuits look a bit exotic. This suggests that, even in the case of a simple circuit model and a simple target function, a good circuit may have an unintuitive form. We will give a detailed description of our circuits in Section 3.

For MAJ_n, we show that for $k = 3, 4$ and 5, there exist a $\Sigma_3^{(+k,-\ell)}$-circuit whose size is smaller than the currently known smallest monotone Σ_3^{+k}-circuit for an appropriately chosen ℓ. Namely, we show that

- **(i)** There exists a $\Sigma_3^{(+3,-6)}$-circuit of size $O(1.2768^n)$ for MAJ_n, which is smaller than the currently known smallest Σ_3^{+3}-circuit of size $O(1.2779^n)$.
- **(ii)** There exists a $\Sigma_3^{(+4,-4)}$-circuit of size $O(1.2040^n)$ for MAJ_n, which is smaller than the currently known smallest Σ_3^{+4}-circuit of size $O(1.2093^n)$.
- **(iii)** There exists a $\Sigma_3^{(+5,-7)}$-circuit of size $O(1.1751^n)$ for MAJ_n, which is smaller than the currently known smallest Σ_3^{+5}-circuit of size $O(1.1760^n)$.

Although the improvement in the size is relatively small, we believe that our results give some hints on how to use negations to speed up the computation of monotone functions in a shallow circuit. Note that, unlike the circuits for IP_n, the circuits for MAJ_n are probabilistic,

i.e., the proof is existential. As a complementary result, we also show that the size of a smallest $\Sigma_3^{(+k,-\infty)}$-circuit for MAJ_n and the size of a smallest a Σ_3^{+k}-circuit for MAJ_n are essentially equal when $k \leq 2$. We also give some non-trivial lower bounds on the size of a $\Sigma_3^{(+k,-\infty)}$-circuit computing MAJ_n for $k \geq 3$. All the results on the complexity of MAJ_n will be given in Section 4.

The constructions of all our circuits are based on a common methodology, which may be of independent interest. First, we obtain a small building block by a computer search, or more specifically by using an IP (integer programming) solver with some additional heuristics in some cases. Then, we use it to construct a circuit for general input size. As expected from the methodology, some circuits are complicated and difficult to explain why these work. All certificates of our circuits are available electronically at `https://gitlab.com/KazAmano/depth-3-circuits`. Note that we use Gurobi Optimizer [14] in our experiments.

1.4 Organization

The rest of the paper is organized as follows. In Section 2, we give preliminaries. In Section 3, we show the construction of depth-three circuits for the inner product functions. In Section 4, we analyze the size of depth-three circuits for the majority function focusing on the effect of negations. Finally, we close the paper with some concluding remarks in Section 5.

2 Preliminaries

For a vector $x \in \{0,1\}^n$, x_i denotes the i-th bit of x and $|x|$ denotes the number of ones in x, i.e., $|x| := \sum_{i=1}^n x_i$. For two vectors $x, y \in \{0,1\}^n$, we write $x \geq y$ if $x_i \geq y_i$ for every $i = 1, \ldots, n$.

As usual, a CNF formula in which each clause contains at most k literals is called a *k-CNF formula*. We also use a terminology *$(+k,-\ell)$-CNF formula* for two non-negative integers k and ℓ that represents a CNF formula in which each clause contains at most k positive literals and at most ℓ negative literals.

In this paper, we concentrate on depth-three OR $\circ$ AND $\circ$ OR circuits consisting of unbounded fan-in AND/OR gates. Each bottom OR gate has positive or negative literals as its input. We consider several subclasses of Σ_3-circuits where the inputs of the bottom gates are restricted.

A *Σ_3^k-circuit* is a Σ_3-circuit in which all inputs of the top OR gate are k-CNF formulas. Similarly, a *$\Sigma_3^{(+k,-\ell)}$-circuit* is a Σ_3-circuit in which all inputs of the top OR gate are $(+k,-\ell)$-CNF formulas. When $\ell = 0$, we simply write this as a Σ^{+k}-circuit. A *$\Sigma_3^{(+k,-\infty)}$-circuit* means a $\Sigma_3^{(+k,-\ell)}$-circuit with an unbounded value of ℓ. A circuit is said to be *monotone* if it does not contain negative literals.

Recall that the *size* of a Σ_3-circuit is defined as the number of *gates* in the circuit. For a Boolean function f, we write the minimum size of a Σ_3^k-circuit that computes f as $s_3^k(f)$. We sometimes consider the minimum fan-in of the top OR gate in a Σ_3^k-circuit that computes f, which is denoted by $\tilde{s}_3^k(f)$. When $k = O(1)$, $s_3^k(f)$ is at most polynomially (in the number of input variables) larger than $\tilde{s}_3^k(f)$. We also use the symbols $\tilde{s}_3^{(+k,-\infty)}$, which is defined analogously to $\tilde{s}_3^k$.

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. For an integer t such that $0 \leq t \leq n$, the number of input vectors $x \in \{0,1\}^n$ with $|x| = t$ such that $f(x) = 1$ is denoted by $|f|_t$. A Boolean function f is said to be *monotone* if $f(x) \geq f(y)$ for every pair of inputs x and y such that $x \geq y$.

Since the top gate of a Σ_3-circuit is an OR gate, every function g that feeds into the top gate satisfies $g^{-1}(1) \subseteq f^{-1}(1)$ when the circuit computes f. We refer to a function g satisfying this condition as being *consistent* with f.

The *inner product mod two function* over n pairs of input variables, denoted by IP_n, is defined as

$$\mathsf{IP}_n(x_1,\ldots,x_n,y_1,\ldots,y_n) := \begin{cases} 1, & \text{if } \sum_{i=1}^n x_i y_i \pmod 2 \equiv 1, \\ 0, & \text{otherwise.} \end{cases}$$

The *majority function* over n input variables, denoted by MAJ_n, is defined as

$$\mathsf{MAJ}_n(x_1,\ldots,x_n) := \begin{cases} 1, & \text{if } \sum_{i=1}^n x_i \geq n/2, \\ 0, & \text{otherwise.} \end{cases}$$

3 Improving Depth-3 Circuits for Inner Product

In this section, we show the upper bounds on the size of Σ_3^2-circuits and Σ_3^3-circuits for IP_n.

▶ **Theorem 1.** *For every sufficiently large n,*

1. $\tilde{s}_3^2(\mathsf{IP}_n) \leq 2^{d_2 n}$ *where* $d_2 = (\log_2 14)/4 = 0.9518\ldots$,
2. $\tilde{s}_3^3(\mathsf{IP}_n) \leq 2^{d_3 n}$ *where* $d_3 = (\log_2 11)/5 = 0.6918\ldots$.

The proof of Theorem 1 consists of two parts. First, we obtain a small Σ_3^k-circuit for IP_m and $\overline{\mathsf{IP}_m}$ for small values of m, where $\overline{\mathsf{IP}_m}$ denotes the negation of IP_m. Then, we use these circuits to construct a circuit computing IP_n for general values of n. The second part relies on the following lemma.

▶ **Lemma 2.** *Let m be some natural number. Suppose that $\tilde{s}_3^k(\mathsf{IP}_m) \leq s_1$ and $\tilde{s}_3^k(\overline{\mathsf{IP}_m}) \leq s_0$. Then, for all n that are multiples of m, $\tilde{s}_3^k(\mathsf{IP}_n) \leq 2^{dn}$ where $d = \log_2(s_0+s_1)/m$.*

Proof. Put $p = n/m$. Let IP_m^0 denote $\overline{\mathsf{IP}_m}$ and IP_m^1 denote IP_m. By the definition of IP_n, it is obvious that

$$\mathsf{IP}_n(x_1,\ldots,x_n,y_1,\ldots,y_n) = \bigvee_{\substack{i_1,\ldots,i_p\in\{0,1\} \\ i_1+\cdots+i_p\equiv 1 \pmod 2}} (\mathsf{IP}_m^{i_1} \wedge \mathsf{IP}_m^{i_2} \wedge \cdots \wedge \mathsf{IP}_m^{i_p}),$$

where we omit the input variables in the RHS of the above equation for simplicity. The t-th IP_m (or $\overline{\mathsf{IP}_m}$) in the brackets gets $(x_{(t-1)m+1},\ldots,x_{tm},y_{(t-1)m+1},\ldots,y_{tm})$.

Since IP_m and $\overline{\mathsf{IP}_m}$ can be represented by the OR of s_1 and s_0 k-CNF formulas, respectively, $(\mathsf{IP}_m^{i_1} \wedge \mathsf{IP}_m^{i_2} \wedge \cdots \wedge \mathsf{IP}_m^{i_p})$ can be represented by the OR of $(\prod_{j=1}^p s_{i_j})$ k-CNF formulas by expansion. Hence, we have

$$\tilde{s}_3^k(\mathsf{IP}_n) \leq \sum_{\substack{i_1,\ldots,i_p\in\{0,1\} \\ i_1+\cdots+i_p\equiv 1 \pmod 2}} \prod_{j=1}^p s_{i_j} < (s_0+s_1)^p = 2^{\log_2(s_0+s_1)n/m}.$$

This completes the proof of Lemma 2. ◀

By Lemma 2, our task now is to find a good Σ_3^k-circuit for IP_m and $\overline{\mathsf{IP}}_m$ for small values of m. In this work, we use an IP (integer programming) solver for this task.

Suppose that the fan-in of the top OR gate is T. We can formulate the condition that the OR of t k-CNF formulas computes the target function f as an integer programming problem.

For each $t = 1,\ldots,T$ and for every possible clause c, we introduce a Boolean variable $F_{t,c}$ that represents whether the clause c is appeared in the t-th CNF formula. For each $t = 1,\ldots,T$ and for every input vector x, we also introduce a Boolean variable $V_{t,x}$ which represents the output of the t-th CNF formula. Obviously, $V_{t,x} = 1$ iff $\sum_{c:c(x)=0} F_{t,c} = 0$ and $V_{t,x} = 0$ iff $\sum_{c:c(x)=0} F_{t,c} \geq 1$. Finally, we impose the additional constraint that $\bigvee_t V_{t,x} = f(x)$ for every x.

Proof of Theorem 1. In order to prove Theorem 1, it is sufficient to verify the following fact by Lemma 2. ◀

▶ **Fact 3.**
(i) $\tilde{s}_3^2(\mathsf{IP}_4) \leq 7$ *and* $\tilde{s}_3^2(\overline{\mathsf{IP}}_4) \leq 7$.
(ii) $\tilde{s}_3^3(\mathsf{IP}_5) \leq 6$ *and* $\tilde{s}_3^3(\overline{\mathsf{IP}}_5) \leq 5$.

Proof. A certificate for the first part of (i) is the OR of the following seven 2-CNF formulas.

```
f1:  (-x2 -y2)(-x3 -y3)(-x4 -y4)(x1)(y1)
f2:  (-x1 -y1)(-x3 -y3)(-x4 -y4)(x2)(y2)
f3:  (-x4 -y4)(x1 -y2)(x2 -y2)(y1 -y2)(-y1 y2)(x3)(y3)
f4:  (-x3 -y3)(x1 -x2)(-x1 x2)(y1 -x2)(-x2 y2)(x4)(y4)
f5:  (-x1 -y1)(-x2 -y2)(-y3 -y4)(x3 -y3)(y3 x4)(y3 y4)
f6:  (-x2 -y2)(x1 -x4)(x3 -y1)(-x1 x4)(-x3 y1)(y1 x4)(y3)(y4)
f7:  (-x1 -y1)(x2 -x4)(x3 -y2)(x3 x2)(-x2 x4)(-x3 y2)(-x3 y3)(y4)
```

Here, for example, `(x2 -y2)` represents a clause $(x_2 \vee \overline{y_2})$.

A certificate for the second part of (i), i.e., for $\overline{\mathsf{IP}}_4$, is the OR of the following seven 2-CNF formulas.

```
f1:  (-x1 -y1)(-x2 -y2)(x3 -x4)(-x3 x4)(y3 -x4)(y4)
f2:  (-x2 -y2)(-x4 -y4)(x1 -y3)(x3 -y1)(-x3 y1)(-x1 y3)
f3:  (-x2 -y2)(-x3 -y3)(x1 -y1)(-y1 x4)(y1 -x4)(-x4 y4)
f4:  (-x3 -y3)(-x4 -y4)(x1 -x2)(-x1 x2)(y1 -x2)(-x2 y2)
f5:  (-x1 -y1)(-x3 -y3)(x2 -y4)(x4 -y4)(y2 -y4)(-y2 y4)
f6:  (-x1 -y1)(-x4 -y4)(-y3 x2)(x3 -y3)(-y3 y2)(y3 -x2)
f7:  (x1)(x2)(x3)(x4)(y1)(y2)(y3)(y4)
```

The correctness of these circuits can be verified by hand or by using a computer. Currently, we do not have a simple explanation of why these circuits work, especially for the first set of formulas.

The certificates for statement (ii), i.e., for IP_5 and $\overline{\mathsf{IP}}_5$, are given in the appendix. ◀

Currently, we do not know whether the bounds in Fact 3 are optimal. Remark that we also observed that $\tilde{s}_3^3(\mathsf{IP}_4) \leq 4$ and $\tilde{s}_3^3(\overline{\mathsf{IP}}_4) \leq 3$, but these yield a weaker bound of $\tilde{s}_3^3(\mathsf{IP}_n) \leq 2^{0.702n}$. We include the certificates for these bounds in the appendix.

4 Negations may Help Depth-3 Circuits computing Majority

In this section, we consider the size of a $\Sigma_3^{(+k,-\ell)}$-circuit computing the majority function for small values of k.

4.1 Motivating Example

When we consider depth-three Σ_3^{+k}-circuits, i.e., an OR of monotone k-CNF formulas, the smallest size of such a circuit for MAJ_n is essentially determined by the maximum of $R_\phi := |\{x \mid \phi(x) = 1 \text{ and } |x| = n/2\}|$ over all monotone k-CNF formulas ϕ consistent with MAJ_n. We write this maximum as R_*.

Precisely, the minimum top fan-in of a Σ_3^{+k}-circuit for MAJ_n is at least $\binom{n}{n/2}/R_*$ and at most $n\binom{n}{n/2}/R_*$. The lower bound is obvious since the top gate is an OR gate. The upper bound can be proved by a standard technique combining random sampling and the union bound (similar to the proof of Theorem 10 in Section 4.3). An alternative proof based on the integral gap of a certain integer programming problem can be found in [17].

Let us consider Σ_3^{+2}-circuits, i.e., a disjunction of monotone 2-CNF formulas. Finding the value of R_* for 2-CNF formulas is closely related to the well-known Turán problem (see e.g., [17]). The value is known to be $R_* = 2^{n/2}$ and the unique extremal formula is given by the disjoint union of $n/2$ edges in an n-vertex graph. Hence the size of a smallest Σ_3^{+2}-circuit for MAJ_n is $\tilde{\Theta}(2^{n/2})$. In Section 4.2, we will verify that the minimum size of a $\Sigma_3^{(+2,-\infty)}$-circuit for MAJ_n is also $\tilde{\Theta}(2^{n/2})$, which means that negations are useless if each bottom gate gets at most two positive literals.

Let us now consider Σ_3^{+3} circuits. To the best of our knowledge, the value of R_* for monotone 3-CNF formulas is unknown. Through our computer experiments, we see that the maximum value of R_ϕ among all n-variable monotone 3-CNF formulas ϕ for $n = 4, 6, 8, 10$ and 12 are $6, 14, 36, 84$ and 216, respectively. Note that, for $n = 4, 8$ and 12, an extremal 3-CNF formula is given by $n/4$ independent copies of 3-uniform hypergraph on four vertices. This suggests that $R_* = 6^{n/4}$, which would imply that the smallest Σ_3^{+3}-circuit for MAJ_n has size $\tilde{\Theta}((2/6^{1/4})^n) = \tilde{\Theta}(1.2778\cdots^n)$. It is tempting to guess that the minimum size of a $\Sigma_3^{(+3,-\infty)}$-circuit is equal to this.

Surprisingly (at least for us), we discovered that this is not true. We found a CNF formula χ on 12 variables in which each clause contains at most three positive literals (and at most four negative literals), such that the value of R_χ is 217, exceeding the maximum value for monotone 3-CNF formulas by one. The description of χ will be given in Section 4.3. As expected, we can use χ to show that there exists a $\Sigma_3^{(+3,-4)}$-circuit of size $\tilde{O}((2/217^{1/12})^n) = O(1.2774^n)$ for MAJ_n. Although the improvement is small, this suggests that negations may be useful for computing MAJ_n by a $\Sigma_3^{(+3,-\infty)}$-circuit.

In the following subsections, we analyze such a phenomenon more carefully.

4.2 Negations are useless for $k \leq 2$

We first show a lower bound on the size a $\Sigma_3^{(+k,-\infty)}$-circuit for MAJ_n. This was essentially shown in [16]. We include the proof here for completeness.

▶ **Theorem 4.** *For every natural number k, $\tilde{s}_3^{(+k,-\infty)}(\mathsf{MAJ}_n) = \Omega(2^n/(k^{n/2}\sqrt{n}))$.*

The proof relies on the notion of the *lower limit* introduced in [16]. Here, for a vector $x \in \{0,1\}^n$ and a set of indices $S \subseteq \{1,\dots,n\}$, $x|_S$ denotes the restriction of x to the set S.

▶ **Definition 5.** *Let $B \subseteq \{0,1\}^n$ be a set of vectors. A vector $y \in \{0,1\}^n$ is a lower k-limit for a set B, if, for any subset of indices $S \subseteq \{1,\dots,n\}$ with $|S| = k$, there exists a vector $x \in B$ such that $x > y$ and $y|_S = x|_S$.* ⌟

▶ **Lemma 6** ([16])**.** *Let $\mathcal{F}$ be a family of s-element subsets of $\{1,\dots,n\}$. If $|\mathcal{F}| > k^s$, then there exists a lower k-limit y for $\mathcal{F}$.* ⌟

Proof of Theorem 4. Let C be any $\Sigma_3^{(+k,-\infty)}$-circuit computing MAJ_n. Let $g_1,\dots,g_m$ be the functions that feed into the top OR gate of C. Note that every g_i is consistent with MAJ_n (i.e., $g_i^{-1}(1) \subseteq \mathsf{MAJ}_n^{-1}(1)$) and that $\bigcup_{i=1}^m g_i^{-1}(1) = \mathsf{MAJ}_n^{-1}(1)$.

For every $i = 1,\dots,m$, let $B_i := \{x \mid x \in g_i^{-1}(1) \wedge |x| = n/2\}$. We claim that $|B_i| \leq k^{n/2}$ for every i, which immediately implies the lemma since $|\mathsf{MAJ}_n^{-1}(1)| = \Omega(2^n/\sqrt{n})$.

The claim can be verified using Lemma 6 as follows. Suppose for the contrary that $|B_i| > k^{n/2}$ for some i. By Lemma 6, there exists a lower k-limit y for B_i. For each clause c in a $(+k,-\infty)$-CNF g_i, let S_c be the set of indices of positive literals that appeared in c. By the definition of the lower limit, there exists a vector $x_c \in B_i$ with $x_c > y$ that coincides with y on S_c, which ensures that $c(y) = c(x_c) = 1$. This holds for every clause c in g_i, which implies that $g_i(y) = 1$. However, it should satisfy that $|y| \leq n/2 - 1$ and hence $y \in \mathsf{MAJ}_n^{-1}(0)$, a contradiction. ◀

▶ **Theorem 7.** *For $k = 1$ and 2, the minimum size of a $\Sigma_3^{(+k,-\infty)}$-circuit for MAJ_n and the minimum size of a Σ_3^{+k}-circuit for MAJ_n are both $\tilde{\Theta}(2^{n/k})$.*

Proof. Both lower bounds are shown by Theorem 4. The upper bound for $k = 1$ is trivial, and the upper bound for $k = 2$ was shown in e.g., [17], as discussed in Section 4.1. ◀

Note that Theorem 4 does not yield a non-trivial lower bound for $k \geq 4$. For such k, we can apply the following theorem that can be proved by a similar argument to the proof of Theorem 4. An explicit value of the lower bounds obtained from Theorem 8 is shown in Table 1 in Section 4.5.

▶ **Theorem 8.** *For every natural number k and for every real number s with $0 < s \leq 0.5$, $\tilde{s}_3^{(+k,-\infty)}(\mathsf{MAJ}_n) = \tilde{\Omega}(2^{dn})$ where $d = (0.5+s)H(s/(0.5+s)) - s\log_2 k$, where $H(\cdot)$ represents the binary entropy function.*

Proof (sketch). The proof is similar to the proof of Theorem 4. Suppose that a $\Sigma_3^{(+k,-\infty)}$-circuit C computes a MAJ_n. Let s be an arbitrary real number with $0 < s \leq 0.5$. We fix arbitrarily chosen $(0.5 - s)n$ input variables to the value 1 in C. The resulting circuit computes the threshold function on $(0.5 + s)n$ variables that output 1 iff the number of ones in an input vector is at least sn.

The rest of the proof is analogous to the proof of Theorem 4. By noticing that the number of vectors $x \in \{0,1\}^{(0.5+s)n}$ with $|x| = sn$ is

$$\binom{(0.5+s)n}{sn} \sim 2^{(0.5+s)nH(s/(0.5+s))},$$

we can complete the proof of Theorem 8 using Lemma 6. ◀

4.3 Negations may be useful for $k \geq 3$

In this subsection, we give some unintuitive construction of depth-three circuits for the majority function using negations. As to the construction for IP_n, we first obtain a good building block by a computer search and then extend it to a circuit for a general input size.

4.3.1 Blow-up Lemma

In the following, we give several lemmas that will be used in the blow-up process. If a "base" function g is monotone, then this step is easy. We can compute MAJ_n by taking an OR of an appropriate number of independent copies of g over random permutations on inputs. Because our base function is not monotone, we need a small twist to this argument.

▶ **Definition 9.** *Suppose that $n \geq 2$ is an even integer. We say that a list of n-variable Boolean functions $(g_{n/2}, g_{n/2+1}, \ldots, g_n)$ satisfies the increasing property, if (i) every g in the list is consistent with MAJ_n (i.e., $g(x) = 0$ for every $x \in \{0,1\}^n$ with $|x| < n/2$), and (ii) for every $t \geq n/2 + 1$, it holds that*

$$|g_t|_t \geq |g_{n/2}|_{n/2} \prod_{m=n/2+1}^{t} \frac{n-m+1}{m-1}.$$

Here, $g_{n/2}, \ldots, g_n$ are not necessarily distinct.

The following theorem gives a probabilistic construction of a depth-three circuit consistent with MAJ_n that outputs 1 on all inputs in the t-th layer of the Boolean cube, when we are given a function g on n variables consistent with MAJ_n such that $|g|_t$ is large.

▶ **Theorem 10.** *Suppose that an n-variable $(+k,-\ell)$-CNF formula g is consistent with MAJ_n. Then, for every $t \geq n/2+1$, there exists a $\Sigma_3^{(+k,-\ell)}$-circuit C of size at most $\frac{\binom{n}{t}}{|g|_t} \ln \binom{n}{t}$ such that (i) $C(x)=0$ for every $x \in \{0,1\}^n$ with $|x|<n/2$ and (ii) $C(x)=1$ for every $x \in \{0,1\}^n$ with $|x|=t$.*

Proof. Let $\mathbf{g}$ be the uniform distribution over all functions obtained from g by permuting the input variables of g.

Let $x \in \{0,1\}^n$ be an arbitrarily fixed input vector that contains t 1's. Then, we have

$$\Pr_{g \in \mathbf{g}}[g(x)=1] = \frac{|g|_t}{\binom{n}{t}}.$$

Let $v := \frac{\binom{n}{t}}{|g|_t} \ln \binom{n}{t}$. Then, we see that

$$\Pr_{g_1,\ldots,g_v \in \mathbf{g}}\left[\bigvee_{i=1}^{v} g_i(x)=0\right] = \left(1-\frac{|g|_t}{\binom{n}{t}}\right)^v < \frac{1}{\binom{n}{t}}.$$

By the union bound, this implies that there are $(+k,-\ell)$-CNF formulas $g_1,\ldots,g_v$ such that $\bigvee_{i=1}^{v} g_i(x)=1$ for every $x \in \{0,1\}^n$ with $|x|=t$. Obviously $\bigvee_{i=1}^{v} g_i(x)=0$ for every $x \in \{0,1\}^n$ with $|x|<n/2$. Hence, $\bigvee_{i=1}^{v} g_i$ gives a desired depth-three circuit, which completes the proof of Theorem 10. ◀

The following lemma intuitively says that we can mimic a monotone function by a list of non-monotone functions with the increasing property.

▶ **Lemma 11.** *Suppose that m is an even positive integer and n is a multiple of m. Let $\mathbf{g}$ be a list of m-variable $(+k,-\ell)$-CNF formulas $(g_{m/2},\ldots,g_m)$ satisfying the increasing property. For each integer t satisfying $n/2 \leq t \leq n$, we define an n-variable Boolean function f_t as*

$$f_t(x_1,\ldots,x_n) := g_{s_1}(x_1,\ldots,x_m) \wedge g_{s_2}(x_{m+1},\ldots,x_{2m}) \wedge \cdots \wedge g_{s_{n/m}}(x_{n-m-1},\ldots,x_n),$$

where $s_i \in \{\lfloor tm/n \rfloor, \lceil tm/n \rceil\}$ for $i=1,\ldots,n/m$ and satisfies $\sum_{i=1}^{n/m} s_i = t$. Then, it holds that

$$\frac{|f_t|_t}{\binom{n}{t}} \geq \frac{|f_{n/2}|_{n/2}}{\binom{n}{n/2}},$$

and $|f_t|_v = 0$ for every integer v satisfying $0 \leq v < n/2$. ⌟

For example, when $n=120$, $m=12$ and $t=63$, f_t is the AND of three g_7's and seven g_6's. The proof of Lemma 11 is postponed to Appendix.

The following theorem is the main body of our blow-up process.

▶ **Theorem 12.** *Let $m \geq 2$ be an even integer. Let $\mathbf{g}=(g_{m/2},\ldots,g_m)$ be a list of m-variable $(+k,-\ell)$-CNF formulas satisfying the increasing property. Then, there exists a $\Sigma_3^{(+k,-\ell)}$-circuit of size at most $O\left(n^{1.5} \cdot \left(\frac{2}{(|g_{m/2}|_{m/2})^{1/m}}\right)^n\right)$ that computes MAJ_n.*

Proof. For each t satisfying $n/2 \leq t \leq n$, we will construct a $\Sigma_3^{(+k,-\ell)}$-circuit C_t of size $O\left(\sqrt{n} \cdot \left(\frac{2}{(|g_{m/2}|_{m/2})^{1/m}}\right)^n\right)$ such that $C_t(x)=1$ for every $x \in \{0,1\}^n$ with $|x|=t$ and $C_t(x)=0$ for every $x \in \{0,1\}^n$ with $|x|<n/2$. By taking the OR of all C_t's, a desired depth-three circuit will be obtained.

For each t, let f_t be a function on n variables defined as in the statement of Lemma 11. Then, by Theorem 10, there exists a $\Sigma_3^{(+k,-\ell)}$-circuit C_t of size at most

$$\frac{\binom{n}{t}}{|f_t|_t} \ln \binom{n}{t} \tag{1}$$

such that $C_t(x) = 1$ for every x with $|x| = t$ and $C_t(x) = 0$ for every x with $|x| < n/2$.

By Lemma 11, Eq. (1) is upper bounded by

$$\frac{\binom{n}{n/2}}{|f|_{n/2}} \ln \binom{n}{n/2} = O\left(\frac{2^n}{\sqrt{n} \cdot |f|_{n/2}} \ln 2^n\right) = O\left(\sqrt{n} \cdot \left(\frac{2}{(|g|_{m/2})^{1/m}}\right)^n\right).$$

This completes the proof of Theorem 12. ◀

4.3.2 Construction for $k = 3$

By the blow-up lemma described in the previous section, what we need is to find a (list of) $(+3, \cdot)$-CNF formula that is consistent with the majority function satisfying the increasing property. For a Boolean function f on n variables, we call a list $(|f|_0, |f|_1, \ldots, |f|_n)$ as a *profile* of f.

▶ **Fact 13.** *There exists a* $(+3, -4)$*-CNF formula on 12 variables that is consistent with* MAJ_{12} *whose profile is* $(0, 0, 0, 0, 0, 0, 217, 394, 363, 196, 66, 12, 1)$*. There also exists a* $(+3, -6)$*-CNF formula on 16 variables that is consistent with* MAJ_{16} *whose profile is* $(0, \ldots, 0, 1314, 2933, 3547, 2710, 1424, 510, 120, 16, 1)$*.*

As to the case of IP_n, we use an IP (integer programming) solver to find these formulas. Essentially, our task is to find a CNF formula g consistent with MAJ_n that maximizes $|g|_{n/2}$. This can easily be formulated by an IP problem as was described in Section 4.2. In certain cases, we impose some additional constraints on the IP problem to narrow the search space. For this reason, most of our base functions have not been shown to be optimal.

Proof. A certificate for the first statement is the AND of the following 26 clauses.

```
(1 2 3 -4)(1 2 4 -3)(1 3 4 -2)(5 6 7 -8)(5 6 8 -7)
(2 3 4)(7 8 9)(7 8 10)
(2 9 10 -3 -4 -5 -6)(3 9 10 -2 -4 -5 -6)(4 9 10 -2 -3 -5 -6)
(5 9 10 -2 -3 -4)(6 9 10 -2 -3 -4)(7 9 10 -1 -5 -6)(8 9 10 -1 -5 -6)
(5 6 11 -9 -10)(5 6 12 -9 -10)(9 10 11 -12)(9 10 12 -11)
(2 11 12 -3 -4 -7 -8)(3 11 12 -2 -4 -7 -8)(4 11 12 -2 -3 -7 -8)
(5 11 12 -1)(6 11 12 -1)(5 11 12 -6)(6 11 12 -5)
```

Here, for example, `(1 2 3 -4)` denotes the clause $(x_1 \vee x_2 \vee x_3 \vee \overline{x_4})$. The verification is easy by using a computer (but not so easy by hand).

A certificate for the second statement, which has 99 clauses, is provided at the aforementioned GitLab repository. ◀

It is easy to check that both profiles satisfy the increasing property. Hence, the second statement of Fact 13 and Theorem 12 immediately imply the following bound.

▶ **Theorem 14.** *There is a* $\Sigma_3^{(+3,-6)}$*-circuit of size* $O(d^n)$ *that computes* MAJ_n*, where* $d = 2/(1314^{1/16}) < 1.2768$*.* ⌟

Here, the values 1314 and 16 in the statement of Theorem 14 represent the eighth element (counting from zero) of the profile and the number of variables in the base function, respectively.

4.3.3 Construction for $k = 4$

▶ **Fact 15.** *There exists a* $(+4, -4)$*-CNF formula on* 10 *variables consistent with* MAJ_{10} *whose profile is* $(0, 0, 0, 0, 0, 160, 120, 120, 45, 10, 1)$.

Proof. Unlike the case of $k = 3$, our certificate for Fact 15 is well-structured.

We show a CNF formula g consisting of two classes of clauses. To simplify the notation, we start the indexing of input variables from 0 instead of 1, i.e., g is a CNF formula over $\{x_0, x_1, \ldots, x_9\}$. The first class consists of $\binom{5}{2}$ clauses all of them are monotone:

$$(x_{2i} \vee x_{2i+1} \vee x_{2j} \vee x_{2j+1}) \qquad (\forall \{i, j\} \subseteq \{0, 1, \ldots, 4\}).$$

The second class consists of $\binom{5}{2} \cdot 2^4 = 80$ clauses each of which contains four positive literals and four negative literals:

$$(x_{2i_1}^{t_1} \vee x_{2i_1+1}^{1-t_1} \vee x_{2i_2}^{t_2} \vee x_{2i_2+1}^{1-t_2} \vee x_{2i_3}^{t_3} \vee x_{2i_3+1}^{1-t_3} \vee x_{2i_4}^{t_4} \vee x_{2i_4+1}^{1-t_4}) \\ (\forall \{i_1, i_2, i_3, i_4\} \subseteq \{0, 1, \ldots, 4\}, \forall \{t_1, t_2, t_3, t_4\} \in \{0, 1\}^4),$$

where x^0 denotes $\overline{x}$ and x^1 denotes x itself. It is not hard to verify that g is consistent with MAJ_{10} and has a profile $(0, 0, 0, 0, 0, 160, 120, 120, 45, 10, 1)$. ◀

Note that the maximum value of $|\phi|_5$ over all monotone 4-CNF formulas ϕ on 10 variables consistent with MAJ_{10} seems to be 136, which is much smaller than $|g|_5 = 160$.

Since $|g|_6$ is not large enough, the function g alone is not sufficient to form a list satisfying the increasing property. We need to introduce another function h on 10 variables. It is a monotone CNF consisting of 20 clauses, each of which contains four variables. The clauses of h are:

```
(0 1 2 4) (0 1 5 8) (0 1 6 9) (0 2 3 9) (0 2 7 8) (0 3 4 5) (0 4 6 7)
(0 5 7 9) (1 2 3 6) (1 2 5 7) (1 3 4 8) (1 4 7 9) (1 6 7 8) (2 3 5 8)
(2 4 5 9) (2 4 6 8) (3 4 6 9) (3 5 6 7) (3 7 8 9) (5 6 8 9).
```

We can see that h has a profile $(0, 0, 0, 0, 0, 132, 190, 120, 45, 10, 1)$. In fact, h is a function that maximizes $|h|_6$ over all monotone 4-CNF formulas on 10 variables that is consistent with MAJ_{10}. Now the list $\mathbf{g} = (g, h, h, h, h, h)$ satisfies the increasing property. Since $2/(|g|_5)^{1/10} = 2/160^{1/10} \sim 1.20398$, we have the following theorem.

▶ **Theorem 16.** *There is a* $\Sigma_3^{(+4,-4)}$*-circuit of size* $O(1.2040^n)$ *that computes* MAJ_n. ⌟

4.3.4 Construction for $k = 5$

For $k = 5$, our best bound relies on a CNF formula with the following property.

▶ **Fact 17.** *There exists a* $(+5, -7)$*-CNF formula on* 16 *variables that is consistent with* MAJ_{16} *whose profile is* $(0, \ldots, 0, 4958, 5312, 4890, 3353, 1820, 560, 120, 16, 1)$. ⌟

We provide a certificate for Fact 17, which has 6817(!) clauses, at the aforementioned GitLab repository. Since this profile satisfies the increasing property, we have the following bound.

▶ **Theorem 18.** *There exists a* $\Sigma_3^{(+5,-7)}$*-circuit of size* $O(d^n)$ *that computes* MAJ_n*, where* $d = 2/(4958^{1/16}) < 1.1751$. ⌟

4.4 Construction based on Covering Design

As we have seen in Introduction, an asymptotically tight bound for the majority function is not known even for monotone Σ_3^k-circuits. It is not hard to show that if we use a monotone k-CNF formula representing $\mathsf{MAJ}_{2(k-1)}$ as a building block, we have a Σ_3^k-circuit of size $\tilde{O}(d^n)$ where $d = 2/\binom{2(k-1)}{k-1}^{1/2(k-1)}$.

One possibility for improvement is to use a combinatorial object called *covering design*, which has a rich history of research (e.g., refer to [5, 7, 13, 24]). A (v, k, t)-*covering design* is a collection of k-element subsets, called blocks, of $\{1, 2, \ldots, v\}$, such that any t-element subset is contained in at least one block. Let $C(v, k, t)$ be the smallest possible number of blocks in a (v, k, t)-covering design.

We can use a covering design as a building block of depth-three circuits for the majority function. Given a $(2k, k, k-1)$-covering design $\mathcal{S}$, let $f_{\mathcal{S}}$ be a monotone k-CNF formula defined as

$$f_{\mathcal{S}} := \bigwedge_{S \in \mathcal{S}} \bigvee_{i \in \overline{S}} x_i,$$

where $\overline{S}$ denotes the set $\{1, 2, \ldots, 2k\} \backslash S$. It is easy to see that $f_{\mathcal{S}}$ is consistent with MAJ_{2k} and satisfies $|f_{\mathcal{S}}|_k = \binom{2k}{k} - |\mathcal{S}|$.

Generally, such a covering design gives a monotone Σ_3^k-circuit of size $\tilde{O}(d^n)$, where

$$d = \left(\binom{2k}{k} - C(2k, k, k-1) \right)^{\frac{1}{2k}}.$$

The size is smaller than the size of a circuit relying on the direct product of k-CNFs formula representing $\mathsf{MAJ}_{2(k-1)}$, if

$$d < \binom{2(k-1)}{k-1}^{\frac{1}{2(k-1)}}. \tag{2}$$

The exact value of $C(2k, k, k-1)$ is known for $k \leq 6$; $C(6, 3, 2) = 6$, $C(8, 4, 3) = 14$, $C(10, 5, 4) = 51$ and $C(12, 6, 5) = 132$ (see e.g., an online database by Gordon [13]). The situation is mixed for this range of k. InEq. (2) holds when $k = 4$ and 6, but it does not hold when $k = 3$ and 5. We think that the problem of determining the values of k that satisfy InEq. (2) is already an intriguing problem.

4.5 Summary

We summarize the upper bounds on the size of a Σ_3-circuit computing MAJ_n in which each bottom gate contains at most k positive literals for $k = 3, 4$ and 5 (see Table 1). In each entry in Table 1, (a,b) represents that the upper bound is $O((2/b^{1/a})^n)$ which is obtained from a base function ϕ on a-variable consistent with MAJ_a satisfying $|\phi^{-1}(1)|_{a/2} = b$. The first line shows the bounds by a direct product of k-CNF formulas representing $\mathsf{MAJ}_{2(k-1)}$. The second line shows the bounds based on a covering design described in Section 4.4. Both circuits are monotone. The third line shows the bounds by our construction using negations. It is very likely that these bounds can further be improved. The lower bounds given by Theorem 8 (for a suitable choice of s found by a numerical calculation) are shown in the last line.

Table 1 The upper bounds on the size of $\Sigma_3^{(+k,-\infty)}$-circuit for $k = 3, 4$ and 5.

	$\Sigma_3^{(+3,-\infty)}$	$\Sigma_3^{(+4,-\infty)}$	$\Sigma_3^{(+5,-\infty)}$
block threshold	$O(1.2779^n)$ (4,6)	$O(1.2140^n)$ (6,20)	$O(1.1760^n)$ (8,70)
covering design	$O(1.2883^n)$ (6,14)	$O(1.2093^n)$ (8,56)	$O(1.1769^n)$ (10,201)
with negations	$O(1.2768^n)$ (16, 1314)	$O(1.2040^n)$ (10,160)	$O(1.1751^n)$ (16,4958)
lower bound	$\Omega(1.2247^n)$	$\Omega(1.1547^n)$	$\Omega(1.1180^n)$

5 Concluding Remark

In this paper, we give some exotic but efficient constructions of Σ_3-circuits for IP_n and MAJ_n. Our construction relies on a computer search. As an outcome of this approach, in some cases, it seems hard to give a simple explanation on why the obtained circuits work. Extracting the reasoning from our circuits would be a good step for further research. Finally, we would like to emphasize that the question of whether an optimal circuit is inherently looking random would be an intriguing challenge.

References

1 Noga Alon and Ravi B. Boppana. The monotone circuit complexity of boolean functions. *Comb.*, 7(1):1–22, 1987. doi:10.1007/BF02579196.

2 Kazuyuki Amano. On the size of depth-two threshold circuits for the inner product mod 2 function. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications – 14th International Conference, LATA 2020*, volume 12038 of *Lecture Notes in Computer Science*, pages 235–247. Springer, 2020. doi:10.1007/978-3-030-40608-0_16.

3 Ravi B. Boppana. Threshold functions and bounded depth monotone circuits. *J. Comput. Syst. Sci.*, 32(2):222–229, 1986. doi:10.1016/0022-0000(86)90027-9.

4 Bruno Pasqualotto Cavalar and Igor C. Oliveira. Constant-depth circuits vs. monotone circuits. In Amnon Ta-Shma, editor, *38th Computational Complexity Conference, CCC 2023, July 17-20, 2023, Warwick, UK*, volume 264 of *LIPIcs*, pages 29:1–29:37. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CCC.2023.29.

5 Jeffrey H. Dinitz Charles J. Colbourn. *Handbook of Combinatorial Designs (Discrete Mathematics and Its Applications)*. Chapman and Hall/CRC, 2006.

6 Vlado Dancík. Complexity of boolean functions over bases with unbounded fan-in gates. *Inf. Process. Lett.*, 57(1):31–34, 1996. doi:10.1016/0020-0190(95)00182-4.

7 Paul Erdős and Joel Spencer. *Probabilistic Methods in Combinatorics*. Academic Press, 1974.

8 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*, pages 89–98. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.19.

9 Jürgen Forster. A linear lower bound on the unbounded error probabilistic communication complexity. *J. Comput. Syst. Sci.*, 65(4):612–625, 2002. doi:10.1016/S0022-0000(02)00019-3.

10 Peter Frankl, Svyatoslav Gryaznov, and Navid Talebanfard. A variant of the vc-dimension with applications to depth-3 circuits. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022*, volume 215 of *LIPIcs*, pages 72:1–72:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.72.

11 Alexander Golovnev, Alexander S. Kulikov, and R. Ryan Williams. Circuit depth reductions. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021*, volume 185 of *LIPIcs*, pages 24:1–24:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ITCS.2021.24.

12 Mika Göös, Ziyi Guan, and Tiberiu Mosnoi. Depth-3 circuits for inner product. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 51:1–51:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.MFCS.2023.51.

13 Daniel M. Gordon. Covering designs. https://www.dmgordon.org/cover/.

14 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: https://www.gurobi.com.

15 András Hajnal, Wolfgang Maass, Pavel Pudlák, Mario Szegedy, and György Turán. Threshold circuits of bounded depth. *J. Comput. Syst. Sci.*, 46(2):129–154, 1993. doi:10.1016/0022-0000(93)90001-D.

16 Johan Håstad, Stasys Jukna, and Pavel Pudlák. Top-down lower bounds for depth-three circuits. *Comput. Complex.*, 5(2):99–112, 1995. doi:10.1007/BF01268140.

17 Shuichi Hirahara. A duality between depth-three formulas and approximation by depth-two. *CoRR*, abs/1705.03588, 2017. arXiv:1705.03588.

18 Daniel M. Kane and Ryan Williams. Super-linear gate and super-quadratic wire lower bounds for depth-two and depth-three threshold circuits. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 633–643. ACM, 2016. doi:10.1145/2897518.2897636.

19 Maria M. Klawe, Wolfgang J. Paul, Nicholas Pippenger, and Mihalis Yannakakis. On monotone formulae with restricted depth (preliminary version). In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, STOC 1984*, pages 480–487. ACM, 1984. doi:10.1145/800057.808717.

20 Jiatu Li and Tianqi Yang. $3.1n - o(n)$ circuit lower bounds for explicit functions. 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 1180–1193. ACM, 2022. doi:10.1145/3519935.3519976.

21 Ramamohan Paturi, Michael E. Saks, and Francis Zane. Exponential lower bounds for depth 3 boolean circuits. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, STOC 1997*, pages 86–91. ACM, 1997. doi:10.1145/258533.258556.

22 W. V. Quine. Two theorems about truth-functions. *Boletín de la Sociedad Matemática Mexicana*, 10(1–2):64–70, 1953.

23 Alexander A. Razborov. On the method of approximations. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, STOC 1989*, pages 167–176. ACM, 1989. doi:10.1145/73007.73023.

24 Vojtech Rödl. On a packing and covering problem. *Eur. J. Comb.*, 6(1):69–78, 1985. doi:10.1016/S0195-6698(85)80023-8.

25 Igor' S. Sergeev. On the complexity of bounded-depth circuits and formulas over the basis of fan-in gates. *Discrete Mathematics and Applications*, 29:241–254, 2019. doi:10.1515/dma-2019-0022.

A Appendix

A.1 Certificates for IP_n

The following is a certificate of $\tilde{s}_3^3(\mathsf{IP}_5) \leq 6$.

```
   f1:  (-x1 -y1 -x2)(-x3 -x4 -y4)(x1 -y1 x2)(x3 x4 -x5)(-x3 -y3 x5)(-x4 -y4
x5)(y1 x2)(-x2 y2)(-x3 y3 -x5)(x3 y4 -x5)(-x5 y5)
   f2:  (-x1 -y1)(-x2 -y2 -y4)(x3 -x5 -y5)(x4 -y4)(-x3 -y3 x5) (y3 -x5 -y5)
(x2 -y2 y4)(y2 y4)(-x3 -y3 y5)
```

```
   f3:  (-x3 -y3)(-x2 -y2 -y5)(x1 -x4 -y4)(-x1 -y1 x4)(x5 -y5)(y1 -x4 -y4)
(-x1 -y1 y4)(x2 y5)(y2 y5)
   f4:  (-y3 -x4 -y4)(-y2 -x5 -y5)(x1)(x2)(-x1 x3)(y1 -x2 -x3)(y2 x5)(-x2 y3
x4)(-x1 y3 y4)(-x2 y2 y5)
   f5:  (-x1 -y1 -x3)(-y3 -x5 -y5)(x2 -x4 -y4)(x3 -x5 -y5)(x1 x3)(-x2 -y2
x4)(-x1 y1 x3)(y2 -x4 -y4)(-x3 y3 x5)(-x2 -y2 y4)(-x3 y3 y5)
   f6:  (x1)(y1)(-x2 -y2 x5)(x2 -x5 -y5)(x3 -x4 -y4)(-x3 -y3 x4)(y2 -x5 -y5)
(y3 -x4 -y4)(-x3 -y3 y4)(-x2 -y2 y5)
```

The following is a certificate of $\tilde{s}_3^3(\overline{\mathsf{IP}_5}) \leq 5$.

```
   f1:  (-x1 -y1)(x2 -x4 -y4)(x3 -x5 -y5)(-x2 -y2 x4)(-x3 -y3 x5)
(y2 -x4 -y4)(y3 -x5 -y5)(-x2 -y2 y4)(-x3 -y3 y5)
   f2:  (-x2 -y2)(x1 -x5 -y5)(x3 -x4 -y4)(-x3 -y3 x4)(-x1 -y1 x5)
(y1 -x5 -y5)(y3 -x4 -y4)(-x3 -y3 y4)(-x1 -y1 y5)
   f3:  (-x3 -y3)(x1 -x2 -y2)(-x1 -y1 x2)(x4 -x5 -y5)(-x4 -y4 x5)
(y1 -x2 -y2)(-x1 -y1 y2)(y4 -x5 -y5)(-x4 -y4 y5)
   f4:  (-x4 -y4)(x1 -x3 -y3)(x2 -x5 -y5)(-x1 -y1 x3)(-x2 -y2 x5)
(y1 -x3 -y3)(y2 -x5 -y5)(-x1 -y1 y3)(-x2 -y2 y5)
   f5:  (-x5 -y5)(x1 -x4 -y4)(x2 -x3 -y3)(-x2 -y2 x3)(-x1 -y1 x4)
(y1 -x4 -y4)(y2 -x3 -y3)(-x2 -y2 y3)(-x1 -y1 y4)
```

The first set of formulas looks quite random, whereas the second set of formulas is well-structured.

The following is a certificate for $\tilde{s}_3^3(\mathsf{IP}_4) \leq 4$.

```
   f1:  (x1)(y1)(-x2 -y2)(x3 -x4 -y4)(-x3 -y3 x4)(y3 -x4 -y4)(-x3 -y3 y4)
   f2:  (x2)(y2)(-x3 -y3)(x1 -x4 -y4)(-x1 -y1 x4)(y1 -x4 -y4)(-x1 -y1 y4)
   f3:  (x3)(y3)(-x4 -y4)(x1 -x2 -y2)(-x1 -y1 x2)(y1 -x2 -y2)(-x1 -y1 y2)
   f4:  (x4)(y4)(-x1 -y1)(x2 -x3 -y3)(-x2 -y2 x3)(y2 -x3 -y3)(-x2 -y2 y3)
```

The following is a certificate for $\tilde{s}_3^3(\overline{\mathsf{IP}_4}) \leq 3$.

```
   f1:  (x1 -x3 -y3)(-x1 -y1 x3)(y1 -x3 -y3)(-x1 -y1 y3)(x2 -x4 -y4)
(-x2 -y2 x4)(y2 -x4 -y4)(-x2 -y2 y4)
   f2:  (x1 -x4 -y4)(-x1 -y1 x4)(y1 -x4 -y4)(-x1 -y1 y4)(x2 -x3 -y3)
(-x2 -y2 x3)(y2 -x3 -y3)(-x2 -y2 y3)
   f3:  (x1 -x2 -y2)(-x1 -y1 x2)(y1 -x2 -y2)(-x1 -y1 y2)(x3 -x4 -y4)
(-x3 -y3 x4)(y3 -x4 -y4)(-x3 -y3 y4)
```

A.2 Proof of Lemma 11

Proof of Lemma 11. We first verify the second part of the statement of Lemma 11, i.e., $|f_v|_t = 0$ for every $v < n/2$. This is almost obvious, since if an input vector $x \in \{0,1\}^n$ contains less than $n/2$ ones, then some function g_{s_i} gets an input containing less than $m/2$ ones, and hence it outputs 0 for x.

We now show the first part of the statement of Lemma 11. To this end, it is sufficient to show that, for every $t \geq n/2 + 1$, it holds that

$$\begin{aligned} |f_t|_t &\geq |f_{n/2}|_{n/2} \cdot \frac{\binom{n}{t}}{\binom{n}{n/2}} \\ &= |f_{n/2}|_{n/2} \prod_{\ell=n/2+1}^{t} \frac{n-\ell+1}{\ell}. \end{aligned} \tag{3}$$

Let $p = n/m$ and $z = \lfloor t/p \rfloor$. Let $\alpha = t \pmod{p}$. Since $\mathbf{g}$ satisfies the increasing property, we have

$$\begin{aligned}
|f_t|_t &\geq (|g_{z+1}|_{z+1})^{\alpha} (|g_z|_z)^{p-\alpha} \\
&\geq (|g_{m/2}|_{m/2})^p \left(\prod_{\ell'=m/2+1}^{z} \frac{m-\ell'+1}{\ell'-1} \right)^p \left(\frac{m-z}{z} \right)^{\alpha} \\
&= |f_{n/2}|_{n/2} \prod_{\ell=n/2+1}^{t} \frac{n-\ell+1}{\ell}.
\end{aligned}$$

This completes the proof of Lemma 11. ◀

Recognizing Unit Multiple Intervals Is Hard

Virginia Ardévol Martínez
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016 Paris, France

Romeo Rizzi
Department of Computer Science, University of Verona, Italy

Florian Sikora
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016 Paris, France

Stéphane Vialette
LIGM, CNRS, Univ Gustave Eiffel, F77454 Marne-la-Vallée, France

Abstract

Multiple interval graphs are a well-known generalization of interval graphs introduced in the 1970s to deal with situations arising naturally in scheduling and allocation. A d-interval is the union of d intervals on the real line, and a graph is a d-interval graph if it is the intersection graph of d-intervals. In particular, it is a unit d-interval graph if it admits a d-interval representation where every interval has unit length.

Whereas it has been known for a long time that recognizing 2-interval graphs and other related classes such as 2-track interval graphs is NP-complete, the complexity of recognizing unit 2-interval graphs remains open. Here, we settle this question by proving that the recognition of unit 2-interval graphs is also NP-complete. Our proof technique uses a completely different approach from the other hardness results of recognizing related classes. Furthermore, we extend the result for unit d-interval graphs for any $d \geqslant 2$, which does not follow directly in graph recognition problems –as an example, it took almost 20 years to close the gap between $d = 2$ and $d > 2$ for the recognition of d-track interval graphs. Our result has several implications, including that recognizing $(x, \ldots, x)$ d-interval graphs and depth r unit 2-interval graphs is NP-complete for every $x \geqslant 11$ and every $r \geqslant 4$.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Mathematics of computing → Graph theory

Keywords and phrases Interval graphs, unit multiple interval graphs, recognition, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.8

Related Version *Full Version*: `https://arxiv.org/abs/2309.11908`

Funding This work was partially supported by the ANR project ANR-21-CE48-0022 ("S-EX-AP-PE-AL").

Acknowledgements Part of this work was conducted when RR was an invited professor at Université Paris-Dauphine.

1 Introduction

Interval graphs are undirected graphs formed from a set of intervals on the real line, with a vertex for each interval and an edge between vertices whose intervals intersect. In particular, they are chordal and perfect graphs. Due to its numerous applications the class of interval graphs is one of the most well-studied classes of graphs [27, 12, 23]. These include DNA mapping [33], resource allocation problems in scheduling theory [1] and ecological niche and food web [6].

The practical applications of interval graphs have led to the study of various generalizations, including multiple interval graphs [22, 29, 16]. A graph is a d-interval graph if each vertex is associated with a d-interval (the union of d disjoint intervals on the real line) instead

of a simple interval, and again, there is an edge between two vertices if and only if the corresponding d-intervals overlap at some point of the real line. This generalization enables us to model more complex situation arising naturally in scheduling and allocation problems, such as multi-task scheduling, allocation of multiple associated linear resources, or transmission of continuous-media data [2]. Applications to bioinformatics, namely to model DNA sequence similarity or RNA secondary structure [19, 30], increased the interest in this class of graphs.

Inside the class of multiple interval graphs, different restrictions have been studied. One of the most natural ones is the subclass of unit d-interval graphs, which corresponds to d-interval graphs that have an interval representation where every interval has unit length. Unit multiple intervals can be applied, for example, to model tasks of the same duration in scheduling.

Apart from their concrete applications, another reason why interval graphs have been widely studied in the literature is because many problems that are NP-hard in general graphs become polynomial-time solvable when restricted to interval graphs: colorability, clique, independent set, or Hamiltonian cycle, to name a few. In particular, recognizing interval graphs is also polynomial, and more precisely, it can be done in linear time [4, 7]. Furthermore, there exist multiple characterizations of interval graphs, including a characterization in terms of forbidden induced subgraphs [21]. This is also the case for unit interval graphs [25], which are exactly graphs that do not contain any claw, tent, net, or induced cycle of length at least 4. Unit interval graphs are also characterized as interval graphs that are claw-free [27].

However, for multiple interval graphs, most problems remain hard, even their recognition, and they do not have any simple characterization. In particular, they are neither chordal graphs nor perfect graphs. It is known that Maximum Clique remains NP-complete in multiple interval graphs, even for unit 2-intervals [13], and so do other problems such as Independent Set or Dominating Set [2, 5]. The parameterized complexity of some of these problems in multiple interval graphs has also been studied, see for instance [18, 10]. With respect to the recognition of multiple interval graphs, it was proven to be NP-hard in 1984 [31]. More precisely, West and Shmoys showed that determining whether the interval number of a graph (i.e., the smallest integer d such that the graph has a disjoint d-interval representation) is smaller or equal to d, for any $d \geqslant 2$, is NP-complete. Furthermore, they also proved that for any $r \geqslant 3$ and any $d \geqslant 2$, determining whether a graph has an r-depth d-interval representation (i.e., a d-interval representation with at most r intervals sharing a common point) is NP-complete. On the other hand, the complexity of recognizing depth 2 d-interval graphs is still open, although it is known to be polynomial for depth 2 unit d-interval graphs [18]. The above-mentioned proof of hardness (for unrestricted depth) was then adapted by Gambette and Vialette for balanced 2-intervals [15], which are 2-interval graphs that admit a representation such that every 2-interval is composed of two intervals of the same length, while intervals of different 2-intervals can have different lengths. In the same paper, the authors also initiate the study of the recognition of unit 2-interval graphs and of (x, x) 2-interval graphs (where the two disjoint open intervals have integer endpoints and have length x), but the complexity of both problems remained unsettled. Note that contrary to the previous characterization by Roberts of unit interval graphs, unit 2-interval graphs *cannot* be characterized as $K_{1,5}$-free 2-interval graphs [28].

Another well-studied generalization of interval graphs are d-track interval graphs, where each vertex is associated to the *union* of d disjoint intervals, each in a different parallel line called *track*. Gyárfás and West proved that their recognition is NP-hard for $d = 2$, and conjectured the same for $d \geqslant 3$ [17]. This conjecture was proven way later in [18] by Jiang, who also showed that recognition remains hard for unit d-track interval graphs for any $d \geqslant 2$, but left the recognition of unit d-interval graphs as an open question.

Multiple track interval graphs can be seen as the union of interval graphs. In the same manner, d-boxicity graphs can be seen as the *intersection* of interval graphs. Boxicity is a graph invariant introduced by Roberts [26] and it is the minimum dimension in which a graph can be represented as the intersection graph of boxes. Furthermore, given a graph $G = (V, E)$, it corresponds to the minimum number of interval graphs on the set of vertices V such that the intersection of their edge sets is G. Their recognition is NP-complete [8, 32], even for $d = 2$ [20].

In this paper, we finally settle the complexity of the recognition of unit 2-interval graphs, answering the open question by Jiang [18]. To do so, we prove that it is NP-hard by reducing from Satisfiability instead of Hamiltonian Path, which has been often used for proving the hardness of the recognition of variants of interval graphs. The reductions from Hamiltonian path in triangle-free cubic graphs used previously to prove the hardness of recognizing d-interval graphs, balanced d-interval graphs and d-track interval graphs all use a special vertex which is adjacent to n vertices of a triangle free graph, and therefore, cannot be directly adapted for unit 2-interval graphs. We then extend the hardness result for unit d-interval graphs, for any $d \geqslant 2$. Note that, as pointed out in the concluding remarks of [18], recognition problems are very different from optimization problems, and the boundary of a graph class is not necessarily harder than that of a subclass[1]. Thus, even though one would expect the recognition of unit d-interval graphs to be hard for any d if it's hard for $d = 2$, it is not directly implied.

Our result has several consequences, namely that recognizing $(x, \ldots, x)$ d-interval graphs and depth r unit d-interval graphs is NP-complete for every $x \geqslant 11$ and every $r \geqslant 4$. Finally, our reduction implies as well a lower bound under the ETH.

Structure of the paper. The paper is organized as follows. Section 2 briefly introduces the necessary concepts and definitions. In Section 3, we present the results of the paper. First, in Subsection 3.1, we prove that a generalization of the recognition of unit 2-intervals, Colored unit 2-interval recognition, is NP-complete. Then, we use this result in Subsection 3.2 to prove the main theorem of the paper, which states the NP-completeness of Unit 2-interval recognition. Finally, we present several implications of our result in Subsection 3.3, namely the NP-completeness of recognizing unit d-interval graphs for every $d \geqslant 2$, and of recognizing $(x, \ldots, x)$ d-interval graphs and depth r unit d-interval graphs for every $x \geqslant 11$ and every $r \geqslant 4$. We conclude with some directions for future work in Section 4.

Due to space constraints, some proofs (marked with $(\star)$) are deferred to the full version of this paper.

2 Definitions

An *interval* is a set of real numbers of the form $[a, b] := \{x \in \mathbb{R} \mid a \leqslant x \leqslant b\}$.[2]

A *d-interval* is the union of d disjoint intervals. A d-interval is *balanced* if all its d intervals have the same length, and *unit* when this common length is 1. A family $\mathcal{F}$ of d-intervals is *balanced* (resp., *unit*) if it comprises only balanced (resp., unit) d-intervals. Notice that,

[1] As an example, the class of $K_{1,5}$-free graphs, which admits a brute-force $\mathcal{O}(n^6)$ time recognition algorithm, contains the class of unit 2-track interval graphs, which is NP-hard to recognize [18].

[2] In the literature, it is not always specified whether the intervals considered for the intersection representation of interval graphs are open or closed. As discussed in [24], the reason for this might be that both definitions lead to the same class of finite graphs [14], even for unit interval graphs. However, note that if we allow the use of both open and closed intervals within one representation, then the class of unit interval graphs obtained is not the same as if we only allowed open or closed intervals within one representation [24].

for $d \geqslant 2$, different d-intervals of a same balanced family may comprise 1-intervals with different lengths. A family $\mathcal{F}$ of d-intervals can be used as a representation of the graph $\Omega(\mathcal{F})$ having the d-intervals of $\mathcal{F}$ as its vertex set, and where two d-intervals are adjacent if and only if their intersection is not empty. A graph G is called a (possibly balanced, unit) *d-interval graph* when it admits a representation $\mathcal{F}$ consisting only of (respectively balanced, unit) d-intervals. Notice that the representing family is not unique (in fact, even only by translating all intervals by a same value, we already obtain an infinite number of them). Multiple interval graphs generalize the standard notion of interval graphs (special case for $d = 1$). In this paper, we will use the term *unit 1-interval* (resp. *unit 1-interval graph*) to denote a classical unit interval (resp. a classical unit interval graph), to avoid confusion with a unit 2-interval (resp. unit 2-interval graphs).

Note that many references do not specify whether the intervals of a d-interval must be disjoint or not, and some even define them as the union of d not necessarily disjoint intervals [29]. However, this might be related to the fact that, when there are no restrictions on the length of the intervals, the two definitions lead to the same class of graphs. This is not true for unit d-intervals, so we study the case where disjointness is required, as in the hardness proof of recognizing multiple interval graphs [31].

A d-interval graph is *proper* when it admits a representing family $\mathcal{F}$ such that no 1-interval is properly contained in another one. The classes of proper and unit 1-interval graphs are equivalent, and they correspond exactly to $K_{1,3}$-free interval graphs. The graph $K_{1,3}$ is the star with 3 leaves, and is also called a *claw*. Equivalently, unit interval graphs are known to be exactly those graphs that do not contain any claw, tent, net, or cycle of length at least 4 as an induced subgraph [25].

A d-interval is a $(x_1, \ldots, x_d)$ d-interval if the d disjoint intervals are open, have integer endpoints, and have lengths $x_1, \ldots, x_d$, respectively.

The *depth* of a family of intervals is the maximum number of intervals that share a common point, and the *representation depth* of a d-interval graph is the minimum depth of any d-interval representation of the graph.

The hierarchy of subclasses of d-interval graphs is as follows [15, 18]: $(x, \ldots, x) \subset (x+1, \ldots, x+1) \subset$ *unit* $\subset$ *balanced* $\subset$ *unrestricted*.

The problem Unit 2-interval recognition is defined as follows.

Unit 2-interval recognition
Input: A graph $G = (V, E)$
Task: Decide whether G has a unit 2-interval representation.

Furthermore, we define a more general version of the above problem, which will be useful to prove the hardness of Unit 2-interval recognition.

Colored unit 2-interval recognition
Input: A graph $G = (V, E)$ and a coloring $\gamma : V \to \{\texttt{white}, \texttt{black}\}$.
Task: Decide whether G has a unit 2-interval representation where:
- each white vertex is represented by a unit 2-interval,
- each black vertex is represented by a unit 1-interval.

We refer to this representation as a *colored unit 2-interval representation*.

3 Hardness of recognizing unit multiple interval graphs

In this section, we prove the main result of this paper, which is the hardness of recognizing unit 2-interval graphs, used later on to prove the hardness of recognizing unit d-intervals for every $d \geqslant 2$. The result for $d = 2$ is obtained in two steps. We first prove that the more

general version Colored unit 2-interval representation is NP-complete, and then reduce this problem to Unit 2-interval recognition, which yields the main result of this paper.

3.1 Hardness of Colored Unit 2-Interval Recognition

Before proceeding to the hardness proof of Colored unit 2-interval recognition, we first introduce the variant of SAT that we will reduce from. In the following, we use the term "j-clause" to refer to a clause that contains exactly j literals.

▶ **Lemma 1** ([11]). (⋆) *Satisfiability is* NP*-complete even when restricted to CNF-formulae such that:*

1. *Every clause contains either* 3 *literals (*3*-clause) or* 2 *literals (*2*-clause).*
2. *Each variable appears in exactly one* 3*-clause.*
3. *Each* 3*-clause is positive monotone, i.e., is comprised of three positive literals.*
4. *Each variable occurs exactly in three clauses, once negated and twice positive.*

We can now proceed to the proof of hardness of Colored Unit 2-Interval Recognition.

▶ **Theorem 2.** *Colored Unit 2-Interval Recognition is* NP*-complete, even for graphs of degree at most 6.*

The rest of the subsection is dedicated to the proof of Theorem 2. We first describe the construction used for the reduction and then prove its correctness.

Construction. Let Ψ be an instance of the variant of SAT described in Lemma 1, formed by a set of Boolean variables $x_1, \ldots, x_n$ and a set of clauses $C_1, \ldots, C_m$. We construct an equivalent instance (G_Ψ, γ_Ψ) of Colored unit 2-interval recognition as follows.

For every variable x_i, we introduce the variable gadget $\hat{V}_i$ (truth setting component), which is the vertex-colored graph on three black vertices A_i, B_i, C_i and three white vertices x_i^1, x_i^2 and x_i^N, with all edges between a black vertex and a white vertex, plus the edges (x_i^1, x_i^2), (C_i, A_i) and (C_i, B_i). We anticipate that the white vertices of $\hat{V}_i$ will be adjacent also to vertices outside $\hat{V}_i$; in order to underline this distinction, these three vertices are called *public*, and the black vertices are called *private*.

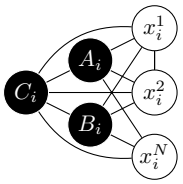


Figure 1 Variable gadget $\hat{V}_i$ corresponding to a variable x_i. Black vertices are displayed with a black background.

Figure 1 illustrates the variable gadget $\hat{V}_i$. Notice that the three white node x_i^1, x_i^2, x_i^N correspond each to precisely one of the occurrences of the represented variable x_i: vertex x_i^N represents the negated occurrence of x_i, vertex x_i^1 represents the positive occurrence in a 3-clause, and vertex x_i^2 represent the positive occurrence in a 2-clause. Therefore, we refer to them as *literal vertices*. Furthermore, note that a vertex of $\hat{V}_i$ is adjacent to A_i if and only if it is adjacent to B_i; and being private, these two nodes will remain false twins also in G. We will exploit this symmetry to simplify the case analysis.

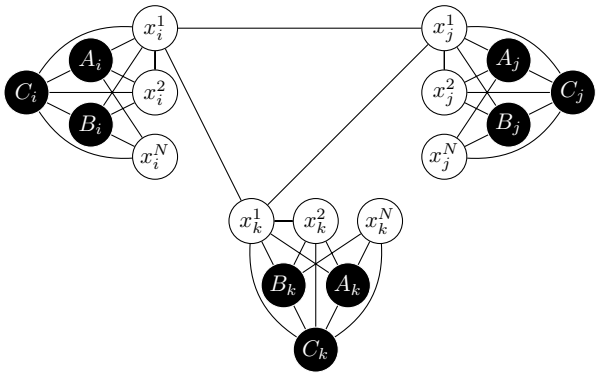


Figure 2 Clause gadget $\hat{C_\alpha}$ associated to a 3-clause $C_\alpha = (x_i \vee x_j \vee x_k)$. Note that in the final graph, each vertex x_i^m, x_j^m, x_k^m, for every $m \in \{1, 2, N\}$, will be incident to exactly 2 edges linking them to vertices outside their variable gadget.

To conclude the construction, we show how to encode each clause C_α, for $\alpha = 1, \ldots, m$. If C_α is a 3-clause, then it is monotone positive, i.e., $C_\alpha = (x_i \vee x_j \vee x_k)$ for some $i, j, k \in \{1, \ldots, n\}$, and all that is needed is to introduce the three edges (x_i^1, x_j^1), (x_j^1, x_k^1), (x_k^1, x_i^1). These three edges comprise the clause gadget (see Figure 2).

If C_α is a 2-clause, say $C_\alpha = (x_i^r \vee x_j^s)$ with $i, j \in \{1, \ldots, n\}$ and $r, s \in \{2, N\}$, then we introduce a public black vertex $L_{i,j}^\alpha$ with a private black neighbor $p_{i,j}^\alpha$ and we add the four edges (x_i^r, x_j^s), $(x_i^r, L_{i,j}^\alpha)$, $(x_j^s, L_{i,j}^\alpha)$ and $(L_{i,j}^\alpha, p_{i,j}^\alpha)$. These four edges together with the two vertices added comprise the clause gadget (see Figure 3).

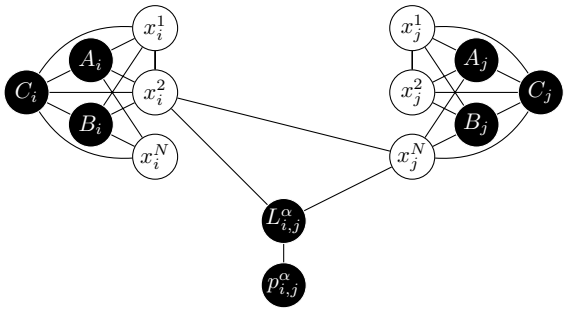


Figure 3 Gadget for a 2-clause $\hat{C_\alpha}$ of the form $C_\alpha = (x_i \vee \overline{x}_j)$.

The description of the reduction is complete. Clearly, G_Ψ has at most $6n + 2m$ vertices and at most $12n + 4m$ edges. We next introduce a few notions to ease the proof that G_Ψ is a colored unit 2-interval graph if and only if Ψ is satisfiable.

▶ **Definition 3.** *Given a colored graph (G, γ), we say that a pair (S, f) formed by a graph S and a function $f : V(S) \mapsto V(G)$ is a split of (G, γ) if f satisfies the following conditions:*

- $|f^{-1}(v)| = 1$ *for every* $v \in V(G)$ *with* $\gamma(v) =$ `black`.
- $|f^{-1}(v)| = 2$ *for every* $v \in V(G)$ *with* $\gamma(v) =$ `white`.
- *For every vertex v of G, $f^{-1}(v)$ is an independent set in S.*
- *For every edge (s, t) of S, $(f(s), f(t))$ is an edge of G.*
- *For every edge (u, v) of G, there exist two vertices s and t in $f^{-1}(\{u, v\})$ such that (s, t) is an edge of S.*

▶ **Definition 4.** *We define the family of splits of G that lead to a unit 1-interval graph as* $\mathcal{S}_{\mathcal{U}}(G) := \{(S, f) \mid (S, f)$ *is a split of* G *and* S *is a unit 1-interval graph*$\}$.

The next lemma shows how a split (S, f) of a colored graph G can be used to certify that G is a colored unit 2-interval graph. This has the advantage of being a truly combinatorial certificate, whereas the number of interval families representing a same graph is infinite with the power of the continuous as soon as at least one exists. Trotter and Harary [29] have already studied vertex splitting in the context of turning a graph into an interval graph.

▶ **Lemma 5.** ($\star$) *A colored graph (G, γ) is a colored unit 2-interval graph if and only if the family $\mathcal{S}_{\mathcal{U}}(G)$ is not empty.*

We can now proceed to study the shape of the possible splits $(S, f) \in \mathcal{S}_{\mathcal{U}}(G_\Psi)$. Let (S, f) be a split of a graph G. For every vertex $v \in V(G)$, we call each element of the set $f^{-1}(v)$ a representative of v. In particular, if v is a white node, we denote its two representatives in $V(S)$ by $f_1^{-1}(v)$ and $f_2^{-1}(v)$. For simplicity, when we refer to an arbitrary representative of a vertex or to the unique representative of a black vertex, we abuse notation and denote it by its label in $V(G)$. Furthermore, given an edge $(u, v) \in G$, we call the edge $(s, t) \in S$, a representative of (u, v) if $s \in f^{-1}(u)$ and $t \in f^{-1}(v)$. Furthermore, given a split (S, f) of the graph G_Ψ, we denote by $S[\hat{V}_i]$ the subgraph of S induced by the vertices of the variable gadget $\hat{V}_i$ (i.e., vertices A_i, B_i, C_i, $f_1^{-1}(x_i^N), f_1^{-1}(x_i^1)$, $f_1^{-1}(x_i^2)$, $f_2^{-1}(x_i^N), f_2^{-1}(x_i^1)$ and $f_2^{-1}(x_i^2)$). Finally, we say that a representative of a literal vertex is an *isolated vertex* if it is not adjacent to any of the private vertices of its variable gadget (i.e., it is not adjacent to A_i, B_i or C_i).

▷ Claim 6. Let (S, f) be an arbitrary graph in $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, none of the black vertices of $S[\hat{V}_i]$ can be adjacent to both representatives of a literal vertex. Furthermore, if a black vertex is adjacent to a representative of x_i^1 and to a representative of x_i^2, these two representatives must be adjacent to each other.

Proof. Suppose that the two representatives of a literal vertex are adjacent to the same black vertex. If the literal vertex is x_i^1 or x_i^2, the black vertex would be a center of a $K_{1,3}$ with these two representatives plus a representative of the vertex x_i^N as leaves. If the literal vertex is x_i^N, the black vertex would be a center of a $K_{1,3}$ with the two representatives of x_i^N and one of x_i^1 or x_i^2 as leaves. Since the graph $K_{1,3}$ is a forbidden induced subgraph for unit 1-interval graphs, this contradicts the fact that S belongs to $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Finally, if a black vertex is adjacent to a representative of x_i^1 and to a representative of x_i^2 which are not adjacent, the black vertex would be a center of a $K_{1,3}$ with these two representatives plus a representative of x_i^N as leaves. ◁

▷ Claim 7. Let (S, f) be an arbitrary split in $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, for every variable x_i with $i \in \{1, \ldots, n\}$, the subgraph $S[\hat{V}_i]$ satisfies at least one of the following two conditions, up to symmetry:

1. The vertex $f_1^{-1}(x_i^N)$ is adjacent to A_i and the vertex $f_2^{-1}(x_i^N)$ is adjacent to B_i.
2. The vertices $f_1^{-1}(x_i^1)$ and $f_1^{-1}(x_i^2)$ are adjacent to each other and to A_i, and the vertices $f_2^{-1}(x_i^1)$ and $f_2^{-1}(x_i^2)$ are adjacent to each other and to B_i.

Proof. By the properties of f, for every edge $(u, v) \in G_\Psi$, there exist elements $s, t \in V(S)$ with $f^{-1}(u) = s$ and $f^{-1}(v) = t$ such that (s, t) is an edge in S.

Suppose condition 1 does not hold, i.e., one of the representatives of x_i^N, say $f_1^{-1}(x_i^N)$, is adjacent to both A_i and B_i. We will show that if condition 2 does not hold either, S cannot be a unit 1-interval graph. Assume that one of the representatives of x_i^1 or x_i^2, say $f_1^{-1}(x_i^1)$

(resp. $f_1^{-1}(x_i^2)$), is adjacent to both A_i and B_i. Then, S contains an induced cycle of length four: $\big(f_1^{-1}(x_i^N), B_i, f_1^{-1}(x_i^1), A_i\big)$ (resp. $\big(f_1^{-1}(x_i^N), B_i, f_1^{-1}(x_i^2), A_i\big)$). This is a forbidden induced subgraph for unit 1-interval graphs, so it contradicts the hypothesis. Thus, it follows that, up to symmetry, vertices $f_1^{-1}(x_i^1)$ and $f_1^{-1}(x_i^2)$ need to be adjacent to A_i, and vertices $f_2^{-1}(x_i^1)$ and $f_2^{-1}(x_i^2)$, to B_i. Finally, by Claim 6, $f_1^{-1}(x_i^1)$ and $f_1^{-1}(x_i^2)$ need to be adjacent to each other, so condition 2 must hold.

Conversely, suppose condition 2 does not hold, i.e., at least one of the representatives of x_i^1 or x_i^2, say $f_1^{-1}(x_i^1)$ w.l.o.g., is adjacent to both A_i and B_i. We will see that condition 1 must hold in order for S to be a unit 1-interval graph. Indeed, if a single representative of x_i^N, say $f_1^{-1}(x_i^N)$, is adjacent to both A_i and B_i, then S contains an induced cycle of size four: $\big(f_1^{-1}(x_i^N), B_i, f_1^{-1}(x_i^1), A_i\big)$. Therefore, one representative of x_i^N must be adjacent to A_i and the other, to B_i. ◁

The previous claim implies that there are four possible configuration of $S[\hat{V}_i]$ such that it does not contain any induced cycles of length greater or equal to 4.

▶ **Lemma 8.** *Let (S, f) be a split of G_Ψ such that $S[\hat{V}_i]$ does not contain any induced cycles of length greater or equal to 4. Then, S satisfies one of the following conditions:*

1. *The vertex $f_1^{-1}(x_i^N)$ is adjacent to A_i and the vertex $f_2^{-1}(x_i^N)$ is adjacent to B_i, while for the rest of the literal vertices, there exists an element in the image via f^{-1} that is an isolated vertex.*
2. *The vertices $f_1^{-1}(x_i^1)$ and $f_1^{-1}(x_i^2)$ are adjacent to each other and to A_i, and the vertices $f_2^{-1}(x_i^1)$ and $f_2^{-1}(x_i^2)$ are adjacent to each other and to B_i, while $f^{-1}(x_i^N)$ contains an isolated vertex.*
3. *The images of x_i^1 and x_i^2 via f^{-1} are as in Case 1 and $f^{-1}(x_i^N)$ is as in Case 2 (see the graph in Figure 4).*
4. *Either the image of x_i^1 or the image of x_i^2 via f^{-1} is as in Case 1 (w.l.o.g., assume it is $f^{-1}(x_i^1)$) so that both representatives of x_i^1 are adjacent to the non-isolated representative of x_i^2; and $f^{-1}(x_i^N)$ is as in Case 2.*

Proof. We have already shown that one of the conditions of Claim 7 must hold. If condition 1 holds, then we have three possible configurations of $f^{-1}(x_i^1)$ and $f^{-1}(x_i^2)$: either both literal vertices have a representative that is isolated (Case 1), only one of them has a representative that is isolated (Case 4), or none of them has an isolated representative (Case 3). On the other hand, if condition 2 holds, the we only have two possible configurations of $f^{-1}(x_i^N)$: one representative of x_i^N is isolated (Case 2), or none of them is (Case 3). Finally, note that in Case 4, both representatives of x_i^1 need to be adjacent to the non-isolated representative of x_i^2 by Claim 6. ◀

The next two claims are devoted to proving that if (S, f) is a split of (G_Ψ, γ) contained in the family $\mathcal{S}_{\mathcal{U}}(G_\Psi)$, then Cases 3 and 4 of Lemma 8 are not possible. To do so, observe that by construction, since every variable appears exactly in three clauses (twice positive and once negated), we know that in G_Ψ, the vertices x_i^N, x_i^1 and x_i^2 all have two incident edges linking them with vertices outside of the variable gadget, called *external edges* in the following. The neighbors outside of the variable gadget are *external vertices*, and they constitute private neighbors of the vertices of the variable gadget, as it is not possible for two different vertices of the variable gadget to be incident to the same external neighbor. We will see that if S is as in Case 3 or Case 4, then the vertices of $S[\hat{V}_i]$ create an induced net with the external neighbors. Since nets are a forbidden induced subgraph for (unit) interval graphs, then S cannot be a unit 1-interval graph.

▷ Claim 9. Let S be an arbitrary graph in $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, for every variable x_i with $i \in \{1, \ldots, n\}$, the subgraph $S[\hat{V}_i]$ cannot be as in Case 3 of Lemma 8.

Proof. Suppose that $S[\hat{V}_i]$ is as in Case 3 of Lemma 8, i.e., as in Figure 4 (where C_i could be in the neighborhood of the other representatives of the vertices, but thanks to the symmetry, these cases are equivalent). We distinguish two cases:

- The two external edges incident to x_i^1 and x_i^2 are incident to two representatives that are adjacent. Then, either $f_1^{-1}(x_i^N), A_i, f_1^{-1}(x_i^1), f_1^{-1}(x_i^2)$, a private neighbor of $f_1^{-1}(x_i^1)$ and a private neighbor of $f_1^{-1}(x_i^2)$ form a net; or $f_2^{-1}(x_i^N), B_i, f_2^{-1}(x_i^1), f_2^{-1}(x_i^2)$, a private neighbor of $f_2^{-1}(x_i^1)$ and a private neighbor of $f_2^{-1}(x_i^2)$ form a net (see the red net in Figure 4).
- Otherwise, at least one of $f_1^{-1}(x_i^1)$ or $f_1^{-1}(x_i^2)$ will be incident to an external edge. Then, $C_i, A_i, f_1^{-1}(x_i^1)$ or $C_i, A_i, f_1^{-1}(x_i^2)$ will create a net together with $B_i, f_1^{-1}(x_i^N)$, and the corresponding external neighbor of $f_1^{-1}(x_i^1)$ or $f_1^{-1}(x_i^2)$, respectively (see the blue net in Figure 4).

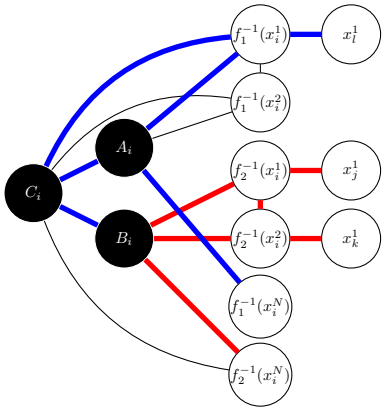


Figure 4 Configuration of $S[\hat{V}_i]$ described in Case 3 of Lemma 8. In red, the net created if both $f_2^{-1}(x_i^1)$ and $f_2^{-1}(x_i^2)$ have an external neighbor. In blue, the net created if $f_1^{-1}(x_i^1)$ has an external neighbor.

In both cases, we have a forbidden induced subgraph for (unit) interval graphs, contradicting the hypothesis that S is a unit interval graph.

◁

▷ Claim 10. (⋆) Let (S, f) be an arbitrary split in $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, for every variable x_i with $i \in \{1, \ldots, n\}$, the subgraph $S[\hat{V}_i]$ cannot be as in Case 4 of Lemma 8.

The proof of Claim 10 uses similar arguments to that of Claim 9 and is thus omitted here. Recall that in Case 1 of Lemma 8, one of the representatives of x_i^1 and one of the representatives of x_i^2 are isolated; and in Case 2 of Lemma 8, one of the representatives of x_i^N is isolated. Therefore, we obtain the following result.

▷ Claim 11. Let (S, f) be an arbitrary split in the family $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, for every variable x_i with $i \in \{1, \ldots, n\}$, the subgraph $S[\hat{V}_i]$ satisfies exactly one of the following two conditions:

1. There is a representative of x_i^1 and a representative of x_i^2 that are isolated vertices (they are either two non-adjacent vertices or they form a K_2).
2. One of the representatives of x_i^N is an isolated vertex.

(a) **(b)**

Figure 5 Representation of the variable gadget associated to the true value (left, 5a) or false value (right, 5b).

Proof. Combining Lemma 8 with Claims 9 and 10, it follows that $S[\hat{V}_i]$ is either as in Case 1 or as in Case 2 of Lemma 8, which means that either one representative of each of x_i^1 and x_i^2 is isolated, or that one representative of x_i^N is isolated, respectively. These options correspond to the interval representations in Figure 5a and Figure 5b, respectively. The reader can check the previous assertion observing the figures, and verify that the external edges incident to each of the vertices x_i^1, x_i^2 and x_i^N can be added in both representations, as we always have either a whole free interval (not depicted in the figures) or one extreme of the interval free for each of the vertices. ◁

The correctness of the reduction now follows from the two lemmas below.

▶ **Lemma 12.** *If Ψ is satisfiable, then the constructed graph $G_\Psi = (V, E)$, $V = V_{white} \cup V_{black}$, admits a colored unit 2-interval representation.*

Proof. Given a satisfying assignment ϕ of Ψ, we explain how to construct a colored unit 2-interval representation of G_Ψ, i.e., a collection of unit 2-intervals $\mathbf{D}_{\texttt{white}} = \{(I_1(v), I_2(v)) \mid v \in V_{\texttt{white}}\}$ and a collection of unit 1-intervals $\mathbf{I}_{\texttt{black}} = \{I_1(v) \mid v \in V_{\texttt{black}}\}$ such that $G \simeq \Omega(\mathbf{D}_{\texttt{white}} \cup \mathbf{I}_{\texttt{black}})$. Note that by Lemma 5, if G_Ψ is a colored unit 2-interval graph, then there exists a split (S, f) in the family $\mathcal{S}_{\mathcal{U}}(G_\Psi)$, and we know how to construct a colored unit 2-interval representation of G_Ψ given a unit 1-interval representation of S by defining the 2-interval associated to a white vertex $v \in V_{\texttt{white}}$ as the union of the interval associated to $f_1^{-1}(v)$ and the interval associated to $f_2^{-1}(v)$; and the 1-interval associated to a black vertex $v \in V_{\texttt{black}}$ as the interval associated to the single vertex $f^{-1}(v)$.

For each variable x_i with $i \in \{1, \ldots, n\}$, if $\Phi(x_i) = true$, we represent the variable gadget $\hat{V}_i$ as shown in Figure 5a, which corresponds exactly to Case 1 of Claim 11. On the other hand, if $\Phi(x_i) = false$, we represent $\hat{V}_i$ as in Figure 5b, which corresponds to Case 2 of Claim 11. Notice that in both representations, the literals that are true have an isolated representative, i.e., one of the intervals associated to them is unused in the representation of $\hat{V}_i$ and remains completely free to display intersections with external neighbors.

After this, it only remains to explain the connections introduced by the clauses.

▷ Claim 13. Given a 3-clause $(x_i \vee x_j \vee x_k)$, there exists a unit interval representation of the subgraph of G_Ψ induced by the vertices of the variable gadgets $\hat{V}_i, \hat{V}_j$ and $\hat{V}_k$.

Proof. Each of the variable gadgets can be represented as in Figure 5a or Figure 5b. To represent the edges associated to the 3-clauses, we first notice that, since the 3-clauses are positive monotone, true literals correspond to true variables. As we are assuming that we have a satisfying assignment, we only have three cases (up to symmetry), which correspond to the three variables being true; exactly two variables being true; and only one variable being true. The literals that are true have a whole free interval to display the intersection, whereas the literals that are false only have the extreme of an interval (while the other extreme is

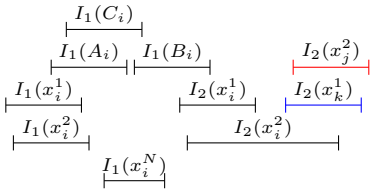


Figure 6 Representation of a 3-clause $(x_i \vee x_j \vee x_k)$, where x_i is set to false while x_j, x_k are set to true.

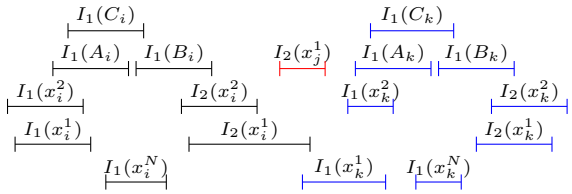


Figure 7 Representation of a 3-clause $(x_i \vee x_j \vee x_k)$, where x_i and x_k are set to false and x_j is set to true.

glued to the rest of the representation of the gadget, see Figure 5b). Let $(x_i \vee x_j \vee x_k)$ be a 3-clause, with $i, j, k \in \{1, \ldots, n\}$. If the three variables are true, we can easily represent the clause by making the three free intervals of the variables – w.l.o.g. $I_2(x_i^1), I_2(x_j^1), I_2(x_k^1)$ – intersect at the same time. On the other hand, if only one variable – say x_i – is false, we can add the two free intervals –$I_2(x_j^1), I_2(x_k^1)$ – to the corresponding extreme of the gadget of the false variable, as in Figure 6. Finally, if two variables are false – say x_i, x_k –, then we need to merge the two interval representations associated to their gadgets and add the free interval – $I_2(x_j^1)$ – in the middle, as in Figure 7. Note that the interval representations given in the figures are not unit, but they are proper, so at the end we will be able to use the algorithm described in [3] to turn it into a unit one. ◁

After representing all the 3-clauses, we can assume that the representations of some of the variable gadgets have been merged two by two (we will never have to merge a gadget more than once since a variable occurs in exactly one 3-clause in Ψ) and we can fix them in the real line separated from one another. The separation between them can be arbitrarily large, and needs to be at least greater than the space needed to place the remaining intervals. The variable gadgets that have not been merged can also be fixed in the real line, while the unused free intervals (corresponding to true literals), the intervals $I_1(L_{i,j}^{\alpha})$, and the intervals $I_1(p_{i,j}^{\alpha})$ remain unplaced.

Now, to display the 2-clauses, we distinguish two cases. First, if both literals are true, then there exists a free interval for each, and we can represent the clause in a separate part of the real line (there is one $L_{i,j}^{\alpha}$ and one $p_{i,j}^{\alpha}$ per clause, so these intervals will never cause a problem). Secondly, if one of the literals is false, then the free interval associated to the true literal needs to be glued to the extreme of the representation of the variable gadget of the false one. Note that there is always one free extreme because the 3-clauses use at most one extreme per variable gadget (and we can extend $I_j(x_i^2)$ to allow the intersection while keeping the representation proper). Note also that we will never need more than two extremes to obtain a representation because, since each variable occurs twice positive and once negated, we can have at most two false literals (when the variable is set to false).

Since we have constructed a proper interval representation, we can now use the algorithm described in [3] to turn the representation into a unit one, as mentioned before. ◀

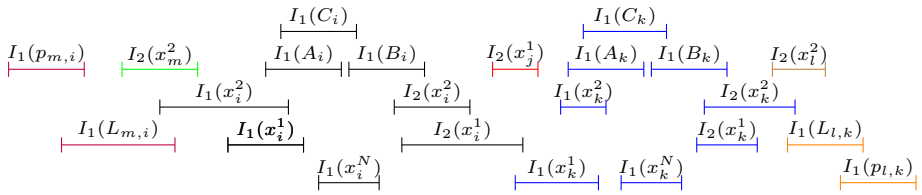


Figure 8 Representation of a longest contiguous block of intervals, where each color represents the intervals associated to a different variable. A longest contiguous block occurs when there is a clause $(x_i \vee x_j \vee x_k)$, where x_i and x_k are set to false and both of them also appear as positive literals in a 2-clause.

Let us now prove the converse implication.

▶ **Lemma 14.** *If the constructed graph $G_\Psi = (V, E)$, $V = V_{white} \cup V_{black}$, admits a colored unit 2-interval representation, then the original formula Ψ is satisfiable.*

Proof. Assume that the constructed graph G_Ψ admits a colored unit 2-interval representation where black vertices are represented by unit 1-intervals and white vertices are represented by unit 2-intervals. As in Claim 11, we study the splits $(S, f) \in \mathcal{S}_\mathcal{U}(G_\Psi)$.

We have already seen in Claim 11 that there are only two possible configurations for $S[\hat{V}_i]$, up to symmetry. Let us assign a truth value to each of the configurations. If $S[\hat{V}_i]$ satisfies condition 1 of Claim 11, we set $\Phi(x_i) = true$. Otherwise, if it satisfies condition 2 of Claim 11, then we set $\Phi(x_i) = false$. Recall that this implies that there is a representative of the vertices representing true literals which remains isolated from its variable gadget.

The following claims restrict the structure of a representable clause gadget. Both use similar arguments, so only the first proof is included here. Given a clause gadget $\hat{C}_\alpha$ in G, we define the clause gadget $S[\hat{C}_\alpha]$ in S as the set of representatives of the edges and vertices of $\hat{C}_\alpha$.

▷ Claim 15. Let (S, f) be an arbitrary split in $\mathcal{S}_\mathcal{U}(G_\Psi)$. Then, for every 3-clause, at least one of the representatives of the literal vertices incident to the clause gadget in S must be an isolated vertex.

Proof. Towards a contradiction, we assume that there exists a 3-clause gadget in S such that none of the representatives of the literal vertices adjacent to the clause gadget are isolated. Let $C_\alpha = x_i \vee x_j \vee x_k$, with $i, j, k \in \{1, \ldots, n\}$ be a (monotone positive) 3-clause. Each of the literal vertices has two external neighbors. In S, either the two external neighbors are incident to the same representative of the literal vertices (and thus only one representative is incident to the clause gadget), or each of them is incident to a different representative. We distinguish two cases, depending on whether only one representative of each literal vertex is incident to the clause gadget, or whether there is at least one literal vertex such that both of its representatives are incident to the clause gadget:

- If only one representative of each literal vertex is incident to the clause gadget in S, then w.l.o.g., the clause gadget is formed by edges $\{(f_1^{-1}(x_i^1), f_1^{-1}(x_j^1)), (f_1^{-1}(x_j^1), f_1^{-1}(x_k^1)), (f_1^{-1}(x_k^1), f_1^{-1}(x_i^1))\}$. By assumption, none of the vertices incident to the clause gadget in S are isolated, so they are all connected to at least one black vertex of their variable gadget. Thus, without loss of generality, $\{f_1^{-1}(x_i^1), f_1^{-1}(x_j^1), f_1^{-1}(x_k^1), A_i, A_j, A_k\}$ form a net (the readers can convince themselves looking at Figure 2). Note that when we say without loss of generality, we are using the symmetry between A_i and B_i.

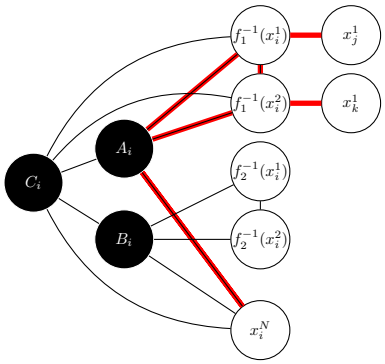


Figure 9 In red, the net created if both representatives of x_i^1 are incident to the clause gadget in S and $f_1^{-1}(x_i^2)$ is incident to an external edge.

- If there is at least one literal vertex such that both of its representatives are incident to the clause gadget, then w.l.o.g., the clause gadget in S contains edges $\{(f_1^{-1}(x_i^1), f_1^{-1}(x_j^1)), (f_1^{-1}(x_k^1), f_2^{-1}(x_i^1))\}$ (and eventually, edges between representatives of x_j^1 and x_k^1). Then, since one of the representative of x_i^2 also has a private neighbor outside of the variable gadget, either the subgraph induced by $\{A_i, f_1^{-1}(x_i^1), f_1^{-1}(x_i^2)\}$ or the subgraph induced by $\{B_i, f_2^{-1}(x_i^1), f_2^{-1}(x_i^2)\}$ (and one private neighbor of each of the three vertices, where the private neighbor of A_i and B_i is x_i^N) is a net. This situation is depicted in Figure 9.

In both cases, the resulting graph S would not be a unit 1-interval graph, contradicting the hypothesis. ◁

▷ Claim 16. ($\star$) Let (S, f) be an arbitrary split in $\mathcal{S}_{\mathcal{U}}(G_\Psi)$. Then, for every 2-clause, at least one of the representatives of the literal vertices incident to the clause gadget in S must be an isolated vertex.

The previous claims imply that there is an isolated literal vertex incident to every 3-clause and to every 2-clause. Since literal vertices that have an isolated representative correspond to true literals in the assignment fixed before, it follows that there is a true literal per clause, and thus, all clauses are satisfied. This finishes the proof of the converse direction. ◀

As the problem is clearly in NP, the polynomial-time construction together with Lemmas 12 and 14 conclude the proof of Theorem 2. The bound on the degree follows because the constructed graph G has maximum degree 6 (the positive literal vertices have degree 4 in the variable gadget and are incident to 2 external edges).

3.2 Hardness of Unit 2-Interval Recognition

We show next that Colored Unit 2-Interval Recognition is polynomial-time reducible to Unit 2-Interval Recognition, which yields the main result of the paper:

▶ **Theorem 17.** *Unit 2-Interval Recognition is* NP*-complete, even for graphs of degree at most 7.*

Proof. We reduce from Colored Unit 2-Interval Recognition, which is NP-hard by Theorem 2. Given any instance (G, γ) of Colored Unit 2-Interval Recognition, where $G = (V, E)$ is a graph and $\gamma : V \to \{\texttt{white}, \texttt{black}\}$ is a vertex-coloring map, we construct an equivalent instance $G' = (V', E')$ of Unit 2-Interval Recognition. Define $n = |V|$ and $V_{\texttt{c}} = \{u \mid u \in V \ \wedge \ \gamma(u) = \texttt{c}\}$ for $c \in \{\texttt{white}, \texttt{black}\}$ (so that $n = |V_{\texttt{white}}| + |V_{\texttt{black}}|$).

We obtain $G' = (V', E')$ from G by replacing every vertex $v \in V_{\texttt{black}}$ by the gadget B_v depicted in Figure 10, which we also call black vertex gadget. Formally, for every $v \in V_{\texttt{black}}$, we add the vertices $V_v = \{a_v^i, b_v^i \mid 0 \leqslant i \leqslant 3\}$ and the edges $E_v = \{(v, a_v^0), (a_v^0, a_v^i), (v, b_v^0), (b_v^0, b_v^i), (a_v^0, b_v^0) \mid 1 \leqslant i \leqslant 3\}$. The gadget B_v is exactly the graph induced by the union of V_v and vertex v. Note that the vertex v of B_v is *public*, that is, it is adjacent to vertices of B_v and to vertices outside of B_v, while the rest of the vertices of B_v are *private*, i.e., they are only adjacent to vertices of B_v.

We have thus constructed a graph G' with vertex set $V' = V \cup \{V_v \mid v \in V_{\texttt{black}}\}$ and edge set $E' = E \cup \{E_v \mid v \in V_{\texttt{black}}\}$. Note that G' contains G as an induced subgraph, as $G'[V] = G$. Combining this with the replacement of every vertex in $V_{\texttt{black}}$ by a gadget with 9 vertices and 9 edges, it follows that $|V'| = |V_{\texttt{white}}| + 9\,|V_{\texttt{black}}|$ and $|E'| = |E| + 9\,|V_{\texttt{black}}|$.

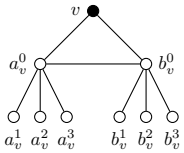


Figure 10 Gadget B_v used to replace every black vertex v of G in the construction of G'. Vertex v is a *public* vertex, as it is adjacent to vertices of the gadget (a_v^0 and b_v^0) and vertices outside the gadget (namely, its neighbors in the original graph G), whereas the rest of the vertices are *private*, as their only neighbors are vertices from the gadget (the ones shown in the figure).

The purpose of the black vertex gadget B_v used to replace every $v \in V_{\texttt{black}}$ in the construction of G' is to restrict the unit 2-interval representations of G'. Indeed, we will see that it forces one of the intervals associated to v to be used exclusively to represent the gadget, while the other interval is used exclusively to represent the rest of the neighborhood of v (which is exactly its neighborhood in the original graph G). Figure 11 shows a unit 2-interval representation $\mathbf{R} = \{(I_1(x), I_2(x)) \mid x \in V_v \cup \{v\}\}$ of B_v such that $I_1(v)$ does not have any points in common with the rest of the intervals of $\mathbf{R}$ (i.e., only $I_2(v)$ is used to represent the gadget). Furthermore, in the given representation, $I_2(v)$ cannot intersect any interval associated to a vertex outside of the gadget, as there is no point of $I_2(v)$ that does not intersect either $I_1(a_v^0)$ or $I_1(b_v^0)$, and both a_v^0 and b_v^0 are private vertices for v. The next claim proves that any unit 2-interval representation of B_v is as in Figure 11, up to symmetry.

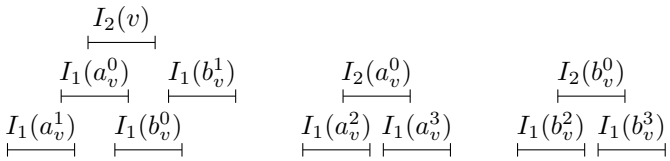


Figure 11 A unit 2-interval representation of B_v (Figure 10), i.e., $\mathbf{D}_{B_v}$ for an arbitrary $v \in V_{\texttt{black}}$. Note that only one interval of v is used ($I_2(v)$), while the other one remains free to display the rest of the neighborhood of v (and is not represented here).

▷ Claim 18. Let $\{(I_1(x), I_2(x)) \mid x \in V_v \cup \{v\}\}$ be a unit 2-interval representation of B_v. Then, there exist some indices $i, j, k \in \{1, 2\}$ such that the representation of $I_i(v), I_j(a_v^0), I_k(b_v^0)$ is contiguous (i.e., the union of the three intervals is an interval) and $I_i(v)$ is properly contained in the union $I_j(a_v^0) \cup I_k(b_v^0)$.

Proof. In the following, we denote an interval associated to a vertex by the name of the vertex if it refers to an arbitrary interval from the corresponding 2-interval (i.e., we will write v to denote $I_1(v)$ or $I_2(v)$ when the choice of interval is irrelevant).

Since a_v^0 and b_v^0 are both centers of an induced $K_{1,4}$, one of the intervals associated to a_v^0, say $I_1(a_v^0)$, needs to intersect v, b_v^0 and one of the a_v^i for some $i \in \{1,2,3\}$, say a_v^1 without loss of generality (because of the symmetry). Furthermore, the intervals of v and b_v^0 that intersect $I_1(a_v^0)$ also need to intersect each other, as otherwise $I_1(a_v^0)$ would intersect three disjoint intervals, contradicting the fact that the representation is unit. On the other hand, $I_2(a_v^0)$ has to intersect the two remaining a_v^i, that is, a_v^2 and a_v^3. Similarly, one of the intervals associated to b_v^0, say $I_1(b_v^0)$, needs to intersect v and a_v^0 (which also intersect each other), and one of the b_v^i for some $i \in \{1,2,3\}$, whereas $I_2(b_v^0)$ intersects the two remaining b_v^i. Again, without loss of generality, we can assume that $I(b_v^0)$ intersects b_v^1 while $I_2(b_v^0)$ intersects b_v^2 and b_v^3.

Thus, we have that $I_1(a_v^0)$ intersects v and $I_1(b_v^0)$ (which also intersect each other), and a_v^1; while $I(b_v^0)$ intersects v and $I_1(a_v^0)$ (which also intersect each other), and b_v^1. This implies that the representation of a_v^1, $I_1(a_v^0)$, b_v^1, $I_1(b_v^0)$ has to be contiguous. Finally, since vertex v is not adjacent to either a_v^1 nor b_v^1, the only possibility to represent the edges (v, a_v^0) and (v, b_v^0) is by placing an interval associated to v, say $I_2(v)$, properly contained in the union $I_1(a_v^0) \cup I_1(b_v^0)$, as in Figure 11. ◁

The next two claims now prove the correctness of the reduction.

▷ Claim 19. If G is a colored unit 2-interval graph, then G' is a unit 2-interval graph.

Proof. Suppose that G is a colored unit 2-interval graph. Then, by assumption, there exists a collection of unit 2-intervals $\mathbf{D}_{\texttt{white}} = \{(I_1(v), I_2(v)) \mid v \in V_{\texttt{white}}\}$ and a collection of unit intervals $\mathbf{I}_{\texttt{black}} = \{I_1(v) \mid v \in V_{\texttt{black}}\}$ such that $G \simeq \Omega\left(\mathbf{D}_{\texttt{white}} \cup \mathbf{I}_{\texttt{black}}\right)$.

From $\mathbf{D} = (\mathbf{D}_{\texttt{white}} \cup \mathbf{I}_{\texttt{black}})$, we show how to construct a unit 2-interval representation $\mathbf{D}'$ of G'. Recall that $(V_{\texttt{white}} \cup V_{\texttt{black}}) = V \subset V'$. Similarly, we will construct $\mathbf{D}'$ such that $\mathbf{D} \subset \mathbf{D}'$. In fact, we will have that $\mathbf{D}' = \mathbf{D} \cup \left(\bigcup_{v \in V_{\texttt{black}}} \mathbf{D}_{B_v}\right)$, where for every $v \in V_{\texttt{black}}$, $\mathbf{D}_{B_v}$ is the interval representation of the gadget B_v. More precisely, we construct $\mathbf{D}'$ as follows:

- For every $v \in V_{\texttt{white}}$, we add to $\mathbf{D}'$ the 2-interval $(I_1(v), I_2(v))$ from $\mathbf{D}$.
- For every $v \in V_{\texttt{black}}$, we add to $\mathbf{D}'$ the interval $I_1(v)$ from $\mathbf{D}$ together with $\mathbf{D}_{B_v}$, i.e., the interval $I_2(v)$ plus the 2-intervals $(I_1(a_v^k), I_2(a_v^k))$ and $(I_1(b_v^k), I_2(b_v^k))$ for $0 \leqslant k \leqslant 3$ as defined in Figure 11.

By construction, $\mathbf{D}'$ is a collection of unit 2-intervals. It is now a simple matter to verify that $G' \simeq \Omega(\mathbf{D}')$. ◁

▷ Claim 20. ($\star$) If G' is a unit 2-interval graph, then G is a colored unit 2-interval graph.

As the problem is clearly in NP, combining the fact that the construction of G' can be carried out in polynomial time with Claims 19 and 20, we obtain that Unit 2-Interval Recognition is NP-complete. The bound on the degree given in the statement of the theorem follows by construction, from adding the black vertex gadgets (Figure 10) to the graph constructed in the proof of Theorem 2 (Figure 2). Indeed, this results in a graph of maximum degree 7, as C_i is adjacent to 5 vertices in the variable gadget and to 2 vertices from the black vertex gadget. ◀

3.3 Consequences and generalizations

We now generalize the result for unit d-interval graphs, with $d \geqslant 2$, which is not directly implied in graph recognition problems, and for some specific cases of unit d-intervals.

▶ **Corollary 21.** ($\star$) *Recognizing unit d-interval graphs is* NP*-complete for every $d \geqslant 2$.*

▶ **Corollary 22.** ($\star$) *Recognizing* $(x, \ldots, x)$ *d-interval graphs is* NP*-complete for every* $x \geqslant 11$ *and every* $d \geqslant 2$.

▶ **Corollary 23.** ($\star$) *Recognizing depth* r *unit d-interval graphs is* NP*-complete for every* $r \geqslant 4$ *and every* $d \geqslant 2$.

The following corollary is based on the Exponential Time Hypothesis (ETH). More details on this notion that we are only touching here can be found in [9, Chapter 14].

▶ **Corollary 24.** ($\star$) *Unless the ETH fails,* Unit d-interval recognition *does not admit an algorithm with running time* $2^{o(|V|+|E|)}$.

4 Concluding remarks

We have proven that recognizing unit d-interval graphs is NP-complete for any $d \geqslant 2$. Furthermore, our reduction implies that recognizing $(x, \ldots, x)$ d-interval graphs for any $x \geqslant 11$, and depth r unit d-interval graphs for any $r \geqslant 4$, is also hard. These results represent a significant step towards settling the landscape of the complexity of the recognition of the different subclasses of d-interval graphs.

However, some questions still remain open. Since we have shown that recognizing depth 4 unit d-interval graphs is NP-complete and it is known that the recognition of depth 2 unit d-interval graphs is polynomial-time solvable [18], it still remains to delineate the exact boundary, i.e., study the case of depth 3 unit d-interval graphs. On the other hand, the complexity of recognizing $(x, \ldots, x)$ d-interval graphs for $x < 11$ is also unknown. Finally, we have obtained a lower bound for the running time of an algorithm for recognizing unit 2-intervals. Since the brute-force algorithm, running in $\mathcal{O}(2^{n^2})$, is far from achieving it, it would be interesting to reduce this gap.

References

1. 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.
2. Reuven Bar-Yehuda, Magnús M. Halldórsson, Joseph Naor, Hadas Shachnai, and Irina Shapira. Scheduling Split Intervals. *SIAM J. Comput.*, 36(1):1–15, 2006. doi:10.1137/S0097539703437843.
3. Kenneth P. Bogart and Douglas B. West. A short proof that 'proper = unit'. *Discret. Math.*, 201(1-3):21–23, 1999. doi:10.1016/S0012-365X(98)00310-0.
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. Ayelet Butman, Danny Hermelin, Moshe Lewenstein, and Dror Rawitz. Optimization problems in multiple-interval graphs. *ACM Trans. Algorithms*, 6(2), 2010. doi:10.1145/1721837.1721856.
6. Joel E. Cohen. *Food Webs and Niche Space*, volume 11 of *Monographs in Population Biology*. Princeton University Press, 1978.
7. Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The LBFS structure and recognition of interval graphs. *SIAM J. Discret. Math.*, 23(4):1905–1953, 2010. doi:10.1137/S0895480100373455.
8. Margaret B. Cozzens. *Higher and Multi-Dimensional Analogues of Interval Graphs*. PhD thesis, Rutgers University, 1982.

9 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.

10 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the Parameterized Complexity of Multiple-Interval Graph Problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.

11 Michael R. Fellows, Jan Kratochvíl, Matthias Middendorf, and Frank Pfeiffer. The complexity of induced minors and related problems. *Algorithmica*, 13(3):266–282, 1995. doi:10.1007/BF01190507.

12 Peter C. Fishburn. *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets*. Wiley, 1985.

13 Mathew C. Francis, Daniel Gonçalves, and Pascal Ochem. The maximum clique problem in multiple interval graphs. *Algorithmica*, 71(4):812–836, 2015. doi:10.1007/s00453-013-9828-6.

14 Peter Frankl and Hiroshi Maehara. Open-interval graphs versus closed-interval graphs. *Discret. Math.*, 63(1):97–100, 1987. doi:10.1016/0012-365X(87)90156-7.

15 Philippe Gambette and Stéphane Vialette. On restrictions of balanced 2-interval graphs. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science, 33rd International Workshop, WG 2007, Dornburg, Germany, June 21-23, 2007. Revised Papers*, volume 4769 of *LNCS*, pages 55–65. Springer, 2007. doi:10.1007/978-3-540-74839-7_6.

16 Jerrold R. Griggs and Douglas B. West. Extremal values of the interval number of a graph. *SIAM J. Algebraic Discret. Methods*, 1(1):1–7, 1980. doi:10.1137/0601001.

17 András Gyárfás and Douglas West. Multitrack interval graphs. *Congressus Numerantium*, pages 109–116, 1995.

18 Minghui Jiang. Recognizing d-interval graphs and d-track interval graphs. *Algorithmica*, 66(3):541–563, 2013. doi:10.1007/s00453-012-9651-5.

19 Deborah Joseph, Joao Meidanis, and Prasoon Tiwari. Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. In Otto Nurmi and Esko Ukkonen, editors, *Algorithm Theory - SWAT '92, Third Scandinavian Workshop on Algorithm Theory, Helsinki, Finland, July 8-10, 1992, Proceedings*, volume 621 of *LNCS*, pages 326–337. Springer, 1992. doi:10.1007/3-540-55706-7_29.

20 Jan Kratochvíl. A special planar satisfiability problem and a consequence of its NP-completeness. *Discret. Appl. Math.*, 52(3):233–252, 1994. doi:10.1016/0166-218X(94)90143-0.

21 Cornelis Gerrit Lekkerkerker and Johan Ch. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51:45–64, 1962.

22 Robert McGuigan. Presentation at NSF-CBMS Conference at Colby College, 1977.

23 Terry A McKee and Fred R McMorris. *Topics in intersection graph theory*. SIAM, 1999.

24 Dieter Rautenbach and Jayme L Szwarcfiter. Unit interval graphs of open and closed intervals. *J. Graph Theory*, 72(4):418–429, 2013. doi:10.1002/jgt.21650.

25 Fred S. Roberts. Indifference graphs. In F. Harary, editor, *Proof Techniques in Graph Theory*, pages 139–146. Academic Press, NY, 1969.

26 Fred S. Roberts. On the boxicity and cubicity of a graph. In W. T. Tutte, editor, *Recent Progress in Combinatorics*, pages 301–310. Academic Press, NY, 1969.

27 Fred S. Roberts. *Graph theory and its applications to problems of society*. SIAM, 1978.

28 Alexandre Simon. Algorithmic study of 2-interval graphs. Master's thesis, Delft University of Technology, 2021.

29 William T. Trotter and Frank Harary. On double and multiple interval graphs. *J. Graph Theory*, 3(3):205–211, 1979. doi:10.1002/jgt.3190030302.

30 Stéphane Vialette. On the computational complexity of 2-interval pattern matching problems. *Theor. Comput. Sci.*, 312(2-3):223–249, 2004. doi:10.1016/j.tcs.2003.08.010.

31 Douglas B. West and David B. Shmoys. Recognizing graphs with fixed interval number is NP-complete. *Disrecte Appl. Math.*, 8:295–305, 1984. doi:10.1016/0166-218X(84)90127-6.

32 Mihalis Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3(3):351–358, 1982.

33 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.

Non-Clairvoyant Makespan Minimization Scheduling with Predictions

Evripidis Bampis
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Alexander Kononov
Sobolev Institute of Mathematics, Novosibirsk, Russia
Novosibirsk State University, Russia

Giorgio Lucarelli
LCOMS, University of Lorraine, Metz, France

Fanny Pascual
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Abstract

We revisit the classical non-clairvoyant problem of scheduling a set of n jobs on a set of m parallel identical machines where the processing time of a job is not known until the job finishes. Our objective is the minimization of the makespan, i.e., the date at which the last job terminates its execution. We adopt the framework of learning-augmented algorithms and we study the question of whether (possibly erroneous) predictions may help design algorithms with a competitive ratio which is good when the prediction is accurate (consistency), deteriorates gradually with respect to the prediction error (smoothness), and not too bad and bounded when the prediction is arbitrarily bad (robustness). We first consider the non-preemptive case and we devise lower bounds, as a function of the error of the prediction, for any deterministic learning-augmented algorithm. Then we analyze a variant of Longest Processing Time first (LPT) algorithm (with and without release dates) and we prove that it is consistent, smooth, and robust. Furthermore, we study the preemptive case and we provide lower bounds for any deterministic algorithm with predictions as a function of the prediction error. Finally, we introduce a variant of the classical Round Robin algorithm (RR), the Predicted Proportional Round Robin algorithm ($PPRR$), which we prove to be consistent, smooth and robust.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases scheduling, online, learning-augmented algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.9

Funding *Evripidis Bampis*: This work was partially supported by the French National Research Agency (Algoridam ANR-19-CE48-0016).
Alexander Kononov: This research was carried out within the framework of the state contract of the Sobolev Institute of Mathematics (project FWNF-2022-0019).

1 Introduction

We consider the problem of scheduling a set of n jobs on m identical machines so that the makespan, i.e., the time when the last job completes its execution, to be minimized. This is one of the most fundamental and well studied problems in scheduling [27, 34, 38]. We focus on the online paradigm of unknown running times where the processing requirement of a job is unknown until the end of its processing (see e.g. [38]), that is the non-clairvoyant setting.

The performance of an online algorithm in the competitive analysis framework is usually evaluated using the competitive ratio [10, 40]. An online algorithm for a minimization problem is ρ-competitive if for every instance of the problem, the value of the objective function of a solution produced by the algorithm is at most ρ times the value of the objective function of an

optimal offline solution. The non-clairvoyant non-preemptive makespan minimization problem on identical machines was the first scheduling problem, and perhaps the first optimization problem, that has been studied using competitive analysis. In 1966, Graham [17] proved that a simple deterministic greedy algorithm, the *List Scheduling algorithm* (LS), has a makespan within a factor of $(2 - 1/m)$ of the makespan of an optimal algorithm with no preemption allowed. The analysis of Graham works also in the non-preemptive case where each job has a release time as well as for the preemptive case, where the processing of any job may be interrupted and resumed at a later time [17, 18]. Given that LS does not use the jobs' processing times, it is also an online non-clairvoyant scheduling algorithm with competitive ratio $(2 - 1/m)$. Shmoys et al. [39] proved that the competitive ratio of any deterministic online algorithm for the non-preemptive non-clairvoyant makespan minimization problem is at least $(2-1/m)$. They also proved that the same tight bound on the competitive ratio holds in the preemptive case. These results showed that in the non-clairvoyant setting there is no difference with respect to the competitive ratio between the preemptive and non-preemptive variants of the makespan minimization problem.

Nevertheless, the assumption of the standard competitive analysis framework that no information is available about the input instance is quite pessimistic. Given the success of Machine Learning methods and Artificial Intelligence in the last years, predictions become available for many optimization problems [2, 33, 44]. However, no guarantees are available concerning the quality of the predictions and several works focus on the following question: "For a given optimization problem and an (unreliable) prediction of the input, is it possible to devise an algorithm with a performance guarantee that is good when the prediction is accurate (consistency), deteriorates gracefully with respect to the prediction error (smoothness), and not too bad and bounded when the prediction is arbitrarily bad (robustness)?" In this paper, we propose to revisit the classical non-clairvoyant makespan minimization scheduling problem in the context of the vibrant area of learning-augmented algorithms that has been formalized in [29] by Lykouris and Vassilvitskii (see [1] for a list of papers in this area) focusing on low complexity learning-augmented algorithms.

2 Further related works

Classical setting. The problem of minimizing the makespan of a schedule of a set of jobs is one of the most fundamental and well studied problems in scheduling theory. As mentioned earlier, Graham proved that LS is a $(2 - \frac{1}{m})$-approximate algorithm [17]. In [18], Graham showed that if the list is ordered in the decreasing order of the processing times of the jobs, then the *Longest Processing Time first* (LPT) algorithm is $(\frac{4}{3} - \frac{1}{3m})$-approximate. Later Coffman et al. [22] proposed a new algorithm, the $MULTIFIT$ algorithm, which leverages the bin packing problem. Improvements to the approximation ratio followed in the works of Friesen [13] and Langston [24, 14]. For a fixed number of machines, a fully polynomial time approximation scheme has been proposed in [37]. Hochbaum and Shmoys [20] proposed a polynomial time approximation scheme for an arbitrary number of processors.For the online (clairvoyant) case with release times, Chen and Vestjens [11] showed that LPT is $\frac{3}{2}$-competitive. When the preemption of jobs is allowed, a simple wrap-around algorithm has been proposed by McNaughton for the offline case [30] which first computes a lower bound of the optimal makespan and then determines a schedule matching this lower bound.

Learning-augmented setting. The framework of learning-augmented algorithms has been formalized by Lykouris and Vassilvitskii [29]. A series of learning-augmented algorithms has been proposed for various problems (see [31, 32]): caching [4, 29, 36, 42], ski-rental [3, 15, 35,

41], routing problems [8, 9, 12, 23], etc. In scheduling, learning-augmented algorithms have been proposed for different criteria, e.g. for the average completion time [7, 21, 28, 35, 43] or the energy consumption in the speed scaling setting [5, 6]. We focus here only on the related work for makespan.

In [25], Lattanzi et al. studied the makespan minimization problem for scheduling a set of jobs with restricted assignments where each job is characterized by a job-dependent subset of the m available machines on which the job can be executed, as well as its processing time. When job j arrives it must be immediately and irrevocably assigned to a machine. They associate a weight to each machine and based on these weights they propose a prediction model where the predicted quantity is the weight of each machine. By using a multiplicative error measure, they show how to obtain a near optimal robust solution for the fractional version based only on the weight-predictions, and use a randomized algorithm for rounding the fractional assignments online with a polylogarithmic loss in the competitive ratio. In [26], Lavastida et al. consider the fractional version of the restricted assignment problem and they prove that the predictions used in [25] can be learned. Moreover, they showed that the predictions are instance-robust. Zhao et al. [45] considered the preemptive-with-restarts makespan minimization problem on a set of uniform parallel machines and they studied it in a learning-augmented setting. They considered an input-prediction model, similar to the one in [35], where for each job a prediction of its processing time is given in advance and they proposed a learning-augmented algorithm with respect to the error prediction that is consistent and robust.

3 Problem Definition, Notations and Preliminaries

In the classical makespan minimization scheduling problem [17], we are given a set $\mathcal{J}$ of n jobs that have to be executed on a set of m parallel identical machines. The execution of each job $j \in \mathcal{J}$ takes a processing time of x_j time units, while it is available for execution only after its release date r_j. Let C_j be the completion time of a job j in a given schedule. The objective is to minimize the completion time of the last job, also known as *makespan* and denoted by $C_{\max} = \max_j\{C_j\}$. We consider both the non-preemptive (Section 4) and the preemptive case (Section 5).

In the non-clairvoyant setting, the *real processing time (or real length)* x_j of each job $j \in \mathcal{J}$ is not known in advance and it becomes known only when j completes its execution. Here, we consider an input-prediction model where for each job $j \in \mathcal{J}$ a prediction of its processing time is given, as it is the case in [21, 35, 45]. Let y_j be the *predicted processing time (or predicted length)* of a job $j \in \mathcal{J}$. We consider the error measure used also in [45].

▶ **Definition 1** (Prediction error, α)**.** *The error of the prediction for job j is defined as $\alpha_j = \max\{\frac{x_j}{y_j}, \frac{y_j}{x_j}\}$ and the error of the prediction is $\alpha = \max_j\{a_j\}$.*

The prediction is perfect if $\alpha = 1$, and in general $\alpha \geq 1$. For example, if $\alpha = 2$ the predicted processing time of a job cannot be more than twice its real processing time or less than half its real processing time. We use the competitive analysis framework to evaluate the performance of the algorithms and our aim is to express the competitive ratio as a function of the prediction error in order to find a trade-off between consistency and robustness.

Given an algorithm $\mathcal{A}$ and an instance I, we denote by $C_{\max}(\mathcal{A}, I)$ the makespan obtained by the algorithm $\mathcal{A}$ on the instance I. In a similar way, we denote by $OPT(I)$ the makespan obtained by the optimal solution on the instance I. Note that the same notation is used for both the classical problem and the problem with predictions, while the optimal solution does not depend on predictions in the latter one. In the case where the instance is clear by the context, we simplify the above notations to $C_{\max}(\mathcal{A})$ and OPT.

Let I be an instance of the classical scheduling problem consisting of n jobs with processing times set to be the real processing times (x_j). Similarly, let I_p be an instance of the problem consisting of the same n jobs with processing times set to be the predicted processing times (y_j). Since (i) the optimal makespan function is monotonic, (ii) the length of each job in I_p is larger than or equal to $\frac{1}{\alpha}$ its real length, and (iii) the length of each job in I_p is smaller than or equal to α times its true length, then the following proposition directly follows.

▶ **Proposition 2.** $\frac{1}{\alpha}OPT(I_p) \leq OPT(I) \leq \alpha OPT(I_p)$

Before continuing with the previously mentioned results, let us present the following example.

▶ **Observation 3.** Consider the following example: we are given $2m-1$ jobs with predicted lengths $y_j = 1$ and m identical machines. Let $x_1 = x_2 = \ldots = x_{m-1} = m-1$, $x_m = \ldots = x_{2m-2} = 1$, and $x_{2m-1} = m$ be the real processing times of the jobs. Hence, $\alpha = m$. For this instance, an optimal solution which knows the real processing times x_j, schedules the job $2m-1$ to a single machine and each of the $m-1$ couples of jobs with processing times 1 and $m-1$ to the remaining $m-1$ machines. Hence, $OPT = m$. However, any deterministic list scheduling algorithm $\mathcal{A}$ (it does not leave any idle time before the starting time of the last scheduled job) cannot take any scheduling decision since it does not know the real processing times, while the known predicted processing times are all identical. Therefore, $\mathcal{A}$ is obliged to create an arbitrary solution which in the worst case will be to schedule the jobs $1, 2, \ldots, m-1$ to the first $m-1$ machines, the jobs $m, m+1, \ldots, 2m-2$ to machine m, and finally the job $2m-1$ to any machine, leading to $C_{\max}(\mathcal{A}) = 2m-1$.

This example shows that any deterministic list scheduling algorithm is at least $(2-\frac{1}{m})$-competitive with predictions. Since any list scheduling algorithm is also at most $(2-\frac{1}{m})$-competitive, we cannot differentiate between different list scheduling algorithms without taking into account some other parameter, such as the value α for example. In what follows, we provide lower and upper bounds as a function of the value of α.

We now formally define the notions of consistency, smoothness and robustness as in [16].

▶ **Definition 4** (Consistency, Smoothness, Robustness)**.** *An algorithm $\mathcal{A}$ is:*

- *ρ_c-consistent, if it is ρ_c-competitive when the prediction is correct, i.e. $\alpha = 1$.*
- *ρ_s-smooth for a continuous function $\rho_s(\alpha)$, if it is $\rho_s(\alpha)$-competitive, where α is the prediction error.*
- *ρ_r-robust, if it is ρ_r-competitive regardless of the prediction error.*

3.1 Our contribution and articulation of the paper

We consider three variants of the problem of scheduling identical machines in the learning-augmented setting. In many works in this area and especially in scheduling (see e.g. [6, 7, 28, 35]), it is common to combine two algorithms, a clairvoyant (assuming predictions to be correct) for consistency and a non-clairvoyant algorithm for robustness. In this work, we propose a single algorithm for each variant (*one stone*) that achieves simultaneously consistency, smoothness and robustness (*for many birds*). In our work, we adapt and analyse two among the most popular algorithms in scheduling, namely LPT and Round Robin (RR), in the learning-augmented framework.

More precisely, for the non-preemptive variants (with and without release dates), we exploit the classical result of Graham [17, 18] which states that LS is a non-clairvoyant $(2-1/m)$-competitive algorithm. This allows us to devise learning-augmented algorithms

that use the predictions in order to create a priority list and then to apply LS. Note that by using such an approach robustness comes for free. In the preemptive case, we devise a new learning-augmented algorithm which is based on the predictions and whose smoothness analysis shows that the competitive ratio gracefully deteriorates with respect to the prediction error from 1 (consistency, when $\alpha = 1$) to $2 - 1/m$ (robustness, when $\alpha \to \infty$). Recall, that for the preemptive case the offline problem can be solved optimally [30] and that for the non-clairvoyant setting, no deterministic algorithm is possible with competitive ratio better than $2 - 1/m$ [39]. In addition, we provide lower bounds for both the non-preemptive and the preemptive variants. Tables 1 and 2 summarize our results with respect to consistency, robustness and smoothness.

The paper is articulated as follows. In Section 4.1, we prove lower bounds for any deterministic learning-augmented algorithm for the identical machines non-clairvoyant non-preemptive makespan minimization problem with predictions. Then, in Section 4.2, we analyze a variant of LPT and we prove that it is consistent, smooth and robust. We also study the non-preemptive problem with release dates and predictions, in Section 4.3, and we show that the generalization of LPT in this setting is also consistent, smooth and robust. Furthermore, we investigate the preemptive case with predictions and we provide lower bounds for any deterministic learning-augmented algorithm as a function of the error (Section 5.1), and then we introduce $PPRR$ that we prove to be consistent, smooth and robust (Section 5.2).

Table 1 Consistency, robustness guarantees.

	Without Predictions		**With Predictions**	
	Competitiveness		*Consistency*	*Robustness*
	Lower Bounds	Upper Bounds	$\alpha = 1$	
Non-preemptive	$2 - 1/m$ [39]	$2 - 1/m$ [17]	4/3	$2 - 1/m$
Non-preemptive with release dates	$2 - 1/m$ [39]	$2 - 1/m$ [17]	3/2	$2 - 1/m$
Preemptive	$2 - 1/m$ [39]	$2 - 1/m$ [17]	1	$2 - 1/m$

4 Non-preemptive Scheduling

In this section, we consider the case with no preemption allowed. We start with two generic lower bounds that hold for any deterministic algorithm, and then we propose and analyze a learning-augmented algorithm based on an adaptation of LPT.

4.1 Lower Bounds

▶ **Proposition 5.** *If $1 \leq \alpha < \sqrt{2}$, there is no deterministic non-clairvoyant algorithm with predictions for scheduling identical machines, with no preemption allowed, which has a competitive ratio smaller than $\frac{1}{2} + \frac{\alpha^2}{2}$.*

Proof. Consider the following instance: m machines, one job of real length α, and m jobs of real length $\frac{1}{\alpha}$. All jobs have predicted length 1. The minimal makespan of a schedule of this instance is $OPT = \max\{\alpha, \frac{2}{\alpha}\} = \frac{2}{\alpha}$ since $\alpha < \sqrt{2}$.

Table 2 Smoothness guarantees.

	Smoothness	
	Lower Bounds	Upper Bounds
Non-preemptive	If $1 \leq \alpha < \sqrt{2}$ $\frac{1}{2} + \frac{\alpha^2}{2}$	$\min\{\frac{2(\alpha^2+1)}{3}, 1 + \frac{\alpha^2}{2}(1 - \frac{1}{m}), 2 - \frac{1}{m}\}$
Non-preemptive with release dates	If $\alpha \geq \sqrt{2}$ $1 + \frac{1}{\lfloor \alpha^2 \rfloor} \left\lfloor \frac{\lfloor \alpha^2 \rfloor (m-1)}{m} \right\rfloor$	$1 + \min\{1, \frac{\alpha^2}{2}\}$
Preemptive	$\frac{m-1}{m} + 1 - \frac{1}{\alpha^2}$ If $\alpha < \sqrt{2}$ $\frac{m\alpha^2+m-1}{\alpha^2+2(m-1)}$ If $\alpha \geq \sqrt{2}$ $\frac{m-1}{m} + \frac{1}{m\lfloor \alpha^2 \rfloor} + 1 - \frac{1}{\lfloor \alpha^2 \rfloor}$	$2 - \frac{\alpha^2+m-2}{\alpha^2 m-1}$

Consider a deterministic algorithm $\mathcal{A}$, and let J be the job scheduled in the last position. Let us assume that J is the job with real processing time α. Therefore, job J starts at a time larger than or equal to $\frac{1}{\alpha}$, and the completion time of J is thus at least $\frac{1}{\alpha} + \alpha$. Therefore, the competitive ratio of $\mathcal{A}$ is at least $\frac{1/\alpha+\alpha}{2/\alpha} = \frac{1}{2} + \frac{\alpha^2}{2}$. ◀

▶ **Proposition 6.** *If $\alpha \geq \sqrt{2}$, there is no deterministic non-clairvoyant algorithm with predictions for scheduling identical machines, with no preemption allowed, which has a competitive ratio smaller than $1 + \frac{1}{\lfloor \alpha^2 \rfloor} \left\lfloor \frac{\lfloor \alpha^2 \rfloor (m-1)}{m} \right\rfloor$.*

Proof. Consider the following instance : m machines, one job of real length α, and $(m-1)\lfloor \alpha^2 \rfloor$ jobs of real length $\frac{\alpha}{\lfloor \alpha^2 \rfloor}$. The optimal makespan of such an instance is $OPT = \alpha$. Let us assume that the predicted length of all the jobs is 1.

Consider a deterministic algorithm $\mathcal{A}$, and let J be the last job to be started in the schedule returned by $\mathcal{A}$. Let us assume that J is the job of real length α. Since J is the last job to be scheduled, it starts at the earliest at time $\left\lfloor \frac{(m-1)\lfloor \alpha^2 \rfloor}{m} \right\rfloor \times \frac{\alpha}{\lfloor \alpha^2 \rfloor}$. Its completion time is thus at least $\left\lfloor \frac{(m-1)\lfloor \alpha^2 \rfloor}{m} \right\rfloor \times \frac{\alpha}{\lfloor \alpha^2 \rfloor} + \alpha$. Since $OPT = \alpha$, the competitive ratio of $\mathcal{A}$ is larger than or equal to $1 + \frac{1}{\lfloor \alpha^2 \rfloor} \left\lfloor \frac{\lfloor \alpha^2 \rfloor (m-1)}{m} \right\rfloor$. ◀

4.2 Common Release Dates

In the case where all jobs are released at time zero, our algorithm works as follows: consider the jobs in non-increasing order of their predicted processing times, i.e. $y_1 \geq y_2 \geq \ldots \geq y_n$. Then, whenever a machine becomes idle, assign to it and schedule non-preemptively the next job according to this order. We call this algorithm the *Longest Predicted Processing Time algorithm* ($LPPT$). Note that each job j finishes x_j units of time after its starting time, while the scheduling decisions are taken based only on the predicted processing times.

In what follows, we establish the following result.

▶ **Theorem 7** (Consistency, Smoothness and Robustness). *LPPT is a non-clairvoyant algorithm with predictions for scheduling identical machines, with no preemption allowed, that achieves a competitive ratio of*

$$\min\{\frac{2(\alpha^2+1)}{3}, 1+\frac{\alpha^2}{2}(1-\frac{1}{m}), 2-\frac{1}{m}\}.$$

Proof. We first give a simple analysis of $LPPT$ for any $\alpha \geq 1$.

▶ **Lemma 8.** *LPPT is a non-clairvoyant algorithm with predictions for scheduling identical machines, with no preemption allowed, that achieves a competitive ratio of*

$$1+\min\{1, \frac{\alpha^2}{2}\}(1-\frac{1}{m}).$$

Proof. Let us consider the schedule returned by $LPPT$ on a given instance I. We assume that the jobs are indexed with respect to the $LPPT$ order, that is in non-increasing order of their predicted processing times: $y_1 \geq y_2 \geq \cdots \geq y_n$. Let t be a job which is completed last (i.e. $C_t = C_{\max}(LPPT)$). We now consider two cases.

Case 1: $y_t > \frac{OPT(I_p)}{2}$. In this case $t \leq m$ and the job t is alone on its machine, and starts at time 0, since otherwise there will be a machine in $OPT(I_p)$ executing two jobs of processing times strictly greater than $\frac{OPT(I_p)}{2}$, which is a contradiction to the value of $OPT(I_p)$. Therefore, $C_{\max} = x_t$, and the schedule is optimal (indeed $OPT(I) \geq x_t$).

Case 2: $y_t \leq \frac{OPT(I_p)}{2}$. By the definition of α, we have that $x_t \leq \alpha y_t \leq \alpha\frac{OPT(I_p)}{2} \leq \alpha^2\frac{OPT(I)}{2}$, where the last inequality holds by Proposition 2. Let s_t be the time at which job t starts to be scheduled. Until s_t, all the machines are busy: this date is thus at most $\frac{\sum_{j\neq t} x_j}{m}$. Since $C_{\max}(LPPT) = s_t + x_t$, we have: $C_{\max}(LPPT) \leq \frac{\sum_{j\neq t} x_j}{m} + x_t = \frac{\sum_j x_j}{m} - \frac{x_t}{m} + x_t$. Since $OPT(I) \geq \frac{\sum_j x_j}{m}$ and $x_t \leq \min\{\frac{\alpha^2 OPT(I)}{2}, OPT(I)\}$, we get: $C_{\max}(LPPT) \leq (1+\min\{1, \frac{\alpha^2}{2}\}(1-\frac{1}{m}))OPT(I)$. ◀

Note, this bound is better than 2 when $\alpha < \sqrt{2}$. Moreover, when the predictions are correct i.e. when $\alpha = 1$, it is $\frac{3}{2} - \frac{1}{2m}$ (whereas LPT is $(\frac{4}{3} - \frac{1}{3m})$-approximate).

We give a better analysis of the $LPPT$ algorithm when $\alpha < \sqrt{2}$.

▶ **Lemma 9.** *When $\alpha < \sqrt{2}$, LPPT is a non-clairvoyant algorithm with predictions for scheduling identical machines, with no preemption allowed, that achieves a competitive ratio of $\frac{2(\alpha^2+1)}{3}$.*

Proof. Let us consider the schedule returned by $LPPT$ on a given instance I. We assume that the jobs are indexed with respect to the $LPPT$ order, that is in non-increasing order of their predicted processing times: $y_1 \geq y_2 \geq \cdots \geq y_n$. Let t be a job which is completed last (i.e. $C_t = C_{\max}(LPPT)$). We now consider two cases.

Case 1: $y_t > \frac{OPT(I_p)}{3}$. If $t \leq m$, then the job t is alone on its machine, and starts at time 0. Therefore, $C_{\max}(LPPT) = x_t$, and the schedule is optimal (indeed $OPT(I) \geq x_t$). Note also that $t \leq 2m$, since otherwise there will be a machine in $OPT(I_p)$ executing three jobs of processing times strictly greater than $\frac{OPT(I_p)}{3}$, which is a contradiction to the value of $OPT(I_p)$. Moreover, we can ignore the jobs $t+1, t+2, \ldots, n$, since the makespan of $LPPT$ is not affected by their removal, while the optimal can only decrease. In what follows in this case, we assume that the instance is reduced to contain only the jobs $1, 2, \ldots, t$.

We next transform the reduced instance I to a new instance I' as follows:

- For $j = 1, 2, \ldots, m$, we set $x'_j = \alpha x_j$.
- For $j = m+1, m+2, \ldots, t$, we set
  $x'_j = \frac{x_j}{\alpha}$, if $x_j > y_j$, and
  $x'_j = x_j$, otherwise.

Recall that we have $\frac{y_j}{\alpha} \leq x_j \leq \alpha y_j$. So, we get $x'_j = \alpha x_j \geq y_j$, for all $j = 1, 2, \ldots, m$, as well as, $x'_j \leq y_j$, for all $j = m+1, m+2, \ldots, t$. It follows that $x'_j \geq x'_{j'}$ for any pair j, j' such that $j \leq m$ and $j' \geq m+1$.

Next, we further modify the instance I' to obtain the instance $\bar{I}$.

- For $j = 1, 2, \ldots, m$, we set $\bar{x}_j = x'_j$.
- For the remaining jobs, we initialize $\mu = m$. In an iterative way and while $\mu < t$, we search for the job $k = \text{argmax}_{\mu+1 \leq j \leq t} x'_j$. Then, for $j = \mu+1, \mu+2, \ldots, k$, we set $\bar{x}_j = x'_k$. We reset $\mu = k$ and pass to the next iteration.

▶ **Property 1.** *Consider a reduced instance I and the corresponding transformed instance $\bar{I}$. Then, the following properties hold.*

(1) *For $j = m+1, m+2, \ldots, t$, we have that $\bar{x}_j \leq \alpha x_j$.*

(2) *$\bar{x}_{m+1} \geq \bar{x}_{m+2} \geq \cdots \geq \bar{x}_t$.*

(3) *For any pair j, j' such that $j \leq m$ and $j' \geq m+1$, we have that $\bar{x}_j \geq \bar{x}_{j'}$.*

Recall that, without loss of generality, we assumed that the instance I is reduced in the first t jobs in non-increasing order of predicted processing times and that $t \leq 2m$. Based on this, we define the algorithm $LPPT_2$ which works like $LPPT$ under the constraint that each machine can execute at most two jobs. It is clear that

$$C_{\max}(LPPT, I) \leq C_{\max}(LPPT_2, I) \tag{1}$$

Similarly, we define the algorithm LPT_2 which works like LPT under the constraint that each machine can execute at most two jobs.

▷ Claim 10. $C_{\max}(LPPT_2, I) \leq \frac{\alpha^2+1}{2\alpha} C_{\max}(LPT_2, \bar{I})$.

▷ Claim 11. $C_{\max}(LPT_2, \bar{I}) \leq \frac{4}{3} OPT(\bar{I})$.

▷ Claim 12. $OPT(\bar{I}) \leq \alpha OPT(I)$.

By combining Equation 1 and Claims 10, 11, 12, we get:

$$\begin{aligned} C_{\max}(LPPT, I) &\leq C_{\max}(LPPT_2, I) \leq \frac{\alpha^2+1}{2\alpha} C_{\max}(LPT_2, \bar{I}) \\ &\leq \frac{2(\alpha^2+1)}{3\alpha} OPT(\bar{I}) \leq \frac{2(\alpha^2+1)}{3} OPT(I). \end{aligned}$$

Case 2: $y_t \leq \frac{OPT(I_p)}{3}$. By the definition of α, we have that $x_t \leq \alpha y_t \leq \alpha \frac{OPT(I_p)}{3} \leq \alpha^2 \frac{OPT(I)}{3}$, where the last inequality holds by Proposition 2. Let s_t be the time at which job t starts to be scheduled. Until s_t, all the machines are busy: this date is thus at most $\frac{\sum_{j \neq t} x_j}{m}$. Since $C_{\max}(LPPT) = s_t + x_t$, we have: $C_{\max}(LPPT) \leq \frac{\sum_{j \neq t} x_j}{m} + x_t = \frac{\sum_j x_j}{m} - \frac{x_t}{m} + x_t$. Since $OPT(I) \geq \frac{\sum_j x_j}{m}$ we get:

$$C_{max}(LPPT) \leq 1 + \frac{\alpha^2}{3}(1 - \frac{1}{m}) OPT(I) \leq \frac{2(\alpha^2+1)}{3}$$ ◀

Lemmas 8 and 9 imply Theorem 7. ◀

▶ Remark 13. For example, if $m = 5$ we have the competitive ratio of $\frac{2(\alpha^2+1)}{3}$ for $\alpha \in [1, \sqrt{\frac{5}{4}}]$, $1 + \frac{\alpha^2}{2}(1 - \frac{1}{m})$ for $\alpha \in [\sqrt{\frac{5}{4}}, \sqrt{2}]$, and $\frac{9}{5}$ for $\alpha > \sqrt{2}$.

4.3 Arbitrary Release Dates

In the case where the jobs have arbitrary release dates, then the algorithm chooses the next *available* job to assign to an idle machine, that is a job which is already released but not yet scheduled. We call this algorithm the *Longest Predicted Processing Time with Release dates algorithm* ($LPPTR$). Here also, each job j finishes x_j units of time after its starting time, while the scheduling decisions are taken based only on the predicted processing times.

▶ **Theorem 14** (Consistency and Smoothness). *$LPPTR$ is a non-clairvoyant algorithm with predictions for scheduling identical machines, with release dates and no preemption allowed, that achieves a competitive ratio of* $1 + \min\{1, \frac{\alpha^2}{2}\}$.

Proof. Let l be a job that is completed last. Let r_l be the release date of job l. If $s_l = r_l$ we have an optimal schedule.

Case 1: $y_l > \frac{OPT(I_p)}{2}$. In the interval between r_l and s_l, no more than one other job is performed. Let $J(r_l)$ be a set of jobs that were performed immediately after the moment of time r_l. Let $Y = \min_{j \in J(r_l)} y_j$. At most m jobs have a processing time greater than $\frac{OPT(I_p)}{2}$. Thus, $Y \leq \frac{OPT(I_p)}{2}$. Hence, $s_l - r_l \leq Y \leq \frac{OPT(I_p)}{2} \leq \frac{\alpha^2 OPT}{2}$. So, we get $C_{\max} = s_l + x_l = s_l - r_l + r_l + x_l = OPT + \frac{\alpha^2 OPT}{2} = OPT(1 + \frac{\alpha^2}{2})$.

Case 2: $y_l \leq \frac{OPT(I_p)}{2}$. Since $OPT(I_p) \leq \alpha OPT$, we have $x_l \leq \alpha y_l \leq \alpha \frac{OPT(I_p)}{2} \leq \alpha^2 \frac{OPT}{2}$. Assume that we have an instance in which $C_{\max} > OPT(1 + \frac{\alpha^2}{2})$. Since $C_{\max} = s_l + x_l$, and since $OPT \geq r_l + x_l$, we have:

$$OPT(1 + \frac{\alpha^2}{2}) < s_l + x_l = s_l - r_l + r_l + x_l \leq s_l - r_l + OPT.$$

We get that $s_l - r_l > \frac{\alpha^2 OPT}{2}$. Let $[t_s, t_f]$ be the last non-empty interval of idle time before the job l begins processing. If such an interval does not exist, then all machines would be busy up to time s_l and $OPT > s_l$. Then, $C_{\max} = s_l + x_l < OPT + \frac{\alpha^2 OPT}{2}$ which is a contradiction, so there exists at least a non-empty interval of idle time before that job l begins.

▶ **Lemma 15.** *In the $LPPTR$ schedule, some jobs begin at or before t_s and complete at or after t_f.*

▶ **Lemma 16.** *Let t_f be the latest point before r_l that some machine is idle. Then $x_l \leq \frac{\alpha^2 OPT}{2} - \frac{t_f}{2}$.*

Additionally we have $OPT \geq s_l - \frac{t_f}{2}$ (see Formula (1.10) in Hochbaum's book [19]). The proof is based on the Lemma 15 and the properties of greedy schedules. Finally, we get $C_{\max} = s_l + x_l \leq s_l + \frac{\alpha^2 OPT}{2} - \frac{t_f}{2} \leq OPT(1 + \frac{\alpha^2}{2})$, contradicting the original assumption on $C_{\max}$. The first inequality follows from Lemma 16. ◀

▶ Remark 17. The fact that $LPPTR$ is $(2 - 1/m)$-robust comes for free from [17, 18] since it is a list scheduling algorithm.

5 Preemptive Scheduling

In this section, we consider the preemptive case and we assume that all jobs and the predictions of their processing times are available at time zero.

5.1 Lower bounds

▶ **Proposition 18.** *If $\alpha \geq \sqrt{2}$, there is no deterministic non-clairvoyant algorithm with predictions for scheduling identical machines, with preemption allowed, which has a competitive ratio smaller than $\frac{m-1}{m} + \frac{1}{m\lfloor\alpha^2\rfloor} + 1 - \frac{1}{\lfloor\alpha^2\rfloor}$.*

▶ **Proposition 19.** *There is no deterministic non-clairvoyant algorithm with predictions for scheduling identical machines, with preemption allowed, which has a competitive ratio smaller than $\frac{m-1}{m} + 1 - \frac{1}{\alpha^2}$.*

▶ **Proposition 20.** *If $\alpha < \sqrt{2}$, there is no deterministic non-clairvoyant algorithm with predictions for scheduling identical machines, with preemption allowed, which has a competitive ratio smaller than $\frac{m\alpha^2+m-1}{\alpha^2+2(m-1)}$.*

5.2 Competitive Algorithm

For this variant, we propose the *Predicted Proportional Round Robin (PPRR) algorithm* which, at each time instant, shares the processing power of the machines to the uncompleted jobs proportionally to their predicted processing times. More specifically, consider a time t. $PPRR$ considers the uncompleted jobs at t in non-increasing order of their predicted processing times, i.e. $y_1 \geq y_2 \geq \ldots \geq y_k$, where k is the number of uncompleted jobs at t. We say that a job i is *mandatory* at time t if $y_i(m-i) \geq \sum_{j=i+1}^{k} y_j$. Each mandatory job is executed alone in a separate machine. Let r be the number of mandatory jobs at time t. Then, a non-mandatory job j at t is executed with speed $\frac{m-r}{\sum_{\ell=r+1}^{k} y_\ell} y_j$. That is, the non-mandatory jobs are executed at a rate proportional to their predicted processing times.

Note that, if $k > m$, then the number of mandatory jobs at t does not exceed $m-1$, while if $k \leq m$, then all k jobs are mandatory. Moreover, we need to recompute the set of mandatory jobs and the speeds of non-mandatory jobs only at time instants corresponding either to the begin of the schedule or to a completion time of a job.

The following lemma shows intuitively that, at a given time t where the total predicted processing time is fixed, the presence of mandatory jobs speeds up the execution of non mandatory jobs.

▶ **Lemma 21.** *Let $\phi(i) = \frac{m-i}{\sum_{\ell=i+1}^{k} y_\ell}$. Consider a r such that $y_i(m-i) \geq \sum_{j=i+1}^{k} y_j$, for each $i = 1, 2, \ldots, r$. For each i, $1 \leq i \leq r$, it holds that $\phi(i-1) \leq \phi(i)$.*

Let us now present some interesting properties of the solution obtained by the $PPRR$ algorithm.

▶ **Property 2.**

(1) *The execution speed of each job can be only increased by the time.*

(2) *If a job becomes mandatory at a time t, then it remains mandatory until its completion.*

(3) *If a job i is mandatory at time t, then any job j such that $j < i$ ($y_j \geq y_i$) is also mandatory.*

(4) *If two jobs i and j do not become mandatory during their execution and $y_i/x_i > y_j/x_j$, then the job i is completed before the job j.*

(5) *If two jobs do not become mandatory during their execution and have the same prediction error α, then they are completed simultaneously.*

(6) *If the prediction is accurate or all jobs have the same prediction error then $PPRR$ is optimal.*

▶ **Theorem 22** (Consistency, Smoothness, Robustness). *$PPRR$ is a non-clairvoyant algorithm with predictions for scheduling identical machines, with preemption allowed, that achieves a competitive ratio of $2 - \frac{\alpha^2+m-2}{\alpha^2 m-1}$. (Hence, $PPRR$ is 1-consistent and $(2-\frac{1}{m})$-robust.)*

Proof. In a schedule constructed by an algorithm $\mathcal{A}$, we call the job i *critical* if $C_i = C_{\max}(\mathcal{A})$. We assume, without loss of generality, that in an optimal schedule, all jobs are completed simultaneously. Indeed, even if this is not the case, then a critical job, say i, is mandatory from time 0 to $OPT = x_i$. Let $X = \sum_j x_j$. We have $x_i > \frac{X}{m}$. We add jobs with a total processing time of $mx_i - X$. The value of the optimum will not change, and the solution of the algorithm with an incorrect prediction can only worsen. Henceforth, we assume that $OPT = X/m$ and in the optimal schedule, all jobs are completed simultaneously.

Let σ^* be an optimal schedule, and σ be the schedule obtained by the algorithm $PPRR$. Moreover, let $s_j(t)$ be the processing speed of a job j at time t in σ. We denote by $x_j(\tau)$ the total execution time of the job j during the interval $[0,\tau]$ in σ. In other words, $x_j(\tau) = \int_0^\tau s_j(t)dt$. Let $\tilde{s}_j(\tau))$ be the average speed of the job j in the interval $[0,\tau]$ in σ, i.e., $\tilde{s}_j(\tau) = x_j(\tau)/\tau$.

Let job c be the critical job in σ, i.e. $C_c = C_{\max}(PPRR)$. Assume that c becomes mandatory at time τ_c in σ. We have

$$C_{\max}(PPRR) = C_c = \tau_c + x_c - x_c(\tau_c) = \tau_c + x_c - \tilde{s}_c(\tau_c)\cdot\tau_c \tag{2}$$

Note that all machines work without any idle time during the interval $[0,\tau_c]$ in σ. It follows that $m\tau_c \le X - (x_c - \tilde{s}_c(\tau_c)\tau_c)$, where $X = \sum_j x_j$. Hence, $\tau_c \le \frac{X-x_c}{m-\tilde{s}_c(\tau_c)}$ and by substituting τ_c in (2), we get

$$C_{\max}(PPRR) \le \frac{(X-x_c)(1-\tilde{s}_c(\tau_c))}{m-\tilde{s}_c(\tau_c)} + x_c \tag{3}$$

Consider the right-hand side of the expression (3) as a function h of $s = \tilde{s}_c(\tau_c)$. Then, we have

$$h'(s) = (X-x_c)\cdot\frac{-(m-s)+(1-s)}{(m-s)^2} = \frac{(X-x_c)(1-m)}{(m-s)^2} < 0$$

Thus, $h(s)$ reaches a maximum when $s = \tilde{s}_c(\tau_c)$ is as small as possible.

In order to get a lower bound to $\tilde{s}_c(\tau_c)$, observe that $s_c(0) = \min\{1, \frac{m-r}{\sum_{\ell=r+1}^n y_\ell} y_c\}$, where r is the number of mandatory jobs at time 0. If $s_c(0) = 1$, then the job c is mandatory starting from time 0, and hence $C_{\max}(PPRR) = x_c$ and $PPRR$ is optimal. In what follows in this proof we consider that $s_c(0) = \frac{m-r}{\sum_{\ell=r+1}^n y_\ell} y_c$. By Lemma 21, we get

$$s_c(0) = \frac{m-r}{\sum_{\ell=r+1}^n y_\ell} y_c \ge \frac{m-0}{\sum_{\ell=0+1}^n y_\ell} y_c = \frac{m}{\sum_{\ell=1}^n y_\ell} y_c = \frac{m y_c}{Y}$$

where $Y = \sum_{\ell=1}^n y_\ell$. By the definition of α, we have that $y_c \ge \frac{x_c}{\alpha}$ and $\alpha X > Y$. So, it holds that $s_c(0) \ge \frac{m x_c}{\alpha^2 X}$. From Property 2(1) of the $PPRR$ algorithm, we have that $s_c(0) \le s_c(t)$ for all $t \in [0,\tau_c]$. Then, it follows that

$$\tilde{s}_c(\tau_c) \ge \frac{m x_c}{\alpha^2 X} \tag{4}$$

By using Equation (4) as a lower bound on $\tilde{s}_c(\tau_c)$ and replacing it in Equation (3) we get

$$C_{\max}(PPRR) \leq \frac{(X-x_c)(1-\frac{mx_c}{\alpha^2 X})}{m-\frac{mx_c}{\alpha^2 X}} + x_c = \frac{(X-x_c)(\alpha^2 X - mx_c)}{\alpha^2 mX - mx_c} + x_c$$

$$= \frac{\alpha^2X^2 - mXx_c - \alpha^2 Xx_c + mx_c^2 + \alpha^2 mXx_c - mx_c^2}{\alpha^2 mX - mx_c}$$

$$= \frac{\alpha^2X^2 - mXx_c - \alpha^2 Xx_c + \alpha^2 mXx_c}{\alpha^2 mX - mx_c} \tag{5}$$

Note that m and α are constants. Fix X and consider the right-hand side of the expression (5) as a function $f(x_c)$. We have

$$f'(x_c) = \frac{(\alpha^2 mX - \alpha^2 X - mX)(\alpha^2 mX - mx_c) + m(\alpha^2X^2 - mXx_c - \alpha^2 Xx_c + \alpha^2 mXx_c)}{(\alpha^2 mX - mx_c)^2}$$

$$= \frac{\alpha^2 mX^2(\alpha^2(m-1) - (m-1))}{(\alpha^2 mX - mx_c)^2} = \frac{\alpha^2 mX^2(\alpha^2-1)(m-1)}{(\alpha^2 mX - mx_c)^2} \geq 0$$

Thus, $f(x_c)$ reaches a maximum when x_c is as large as possible. By our initial observation, we have that $x_c \leq X/m$ and by replacing in Equation (5) we get

$$C_{\max}(PPRR) \leq \frac{\alpha^2X^2 - X^2 - \alpha^2X^2/m + \alpha^2X^2}{\alpha^2 mX - X}$$

Observe that in an optimal schedule the total execution load is equally partitioned to all machines, and hence $OPT \geq \frac{X}{m}$. Therefore, for the competitive ratio ρ of $PPRR$ we have

$$\rho \leq \frac{\alpha^2X^2 - X^2 - \alpha^2X^2/m + \alpha^2X^2}{\alpha^2X^2 - X^2/m} = \frac{2\alpha^2 m - m - \alpha^2}{\alpha^2 m - 1} = 2 - \frac{\alpha^2 + m - 2}{\alpha^2 m - 1}.$$

The consistency (resp. robustness) ratio is achieved by replacing $\alpha = 1$ (resp. taking the bound when $\alpha \to \infty$). ◀

Figure 1 shows the competitive ratio of algorithm PPRR as well as lower bounds on the ratio of any deterministic preemptive algorithm. As we can see, lower bounds and upper bounds are quite close, and get closer when m increases.

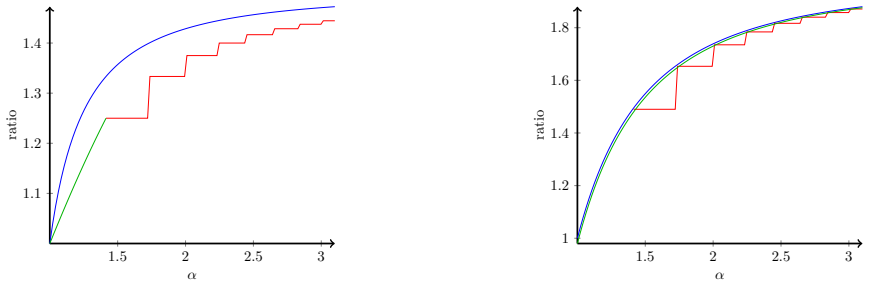


Figure 1 In blue: competitive ratio of algorithm PPRR as a function of α (x axis). In green and red: lower bounds on the competitive ratio of a deterministic algorithm with preemption. In red: lower bound given by Proposition 18. Left (resp. Right): ratio when $m = 2$ (resp. $m = 50$). On the left, in green: lower bound given by Proposition 20 (the bound of Proposition 19 is not drawn here since when $m = 2$, it is lower than the other lower bounds). On the right, in green: lower bound given by Proposition 19 (the bound of Proposition 20 is not drawn here since when $m = 50$, it is lower than the other lower bounds).

References

1. https://algorithms-with-predictions.github.io.
2. Maryam Amiri and Leili Mohammad Khanli. Survey on prediction models of applications for resources provisioning in cloud. *J. Netw. Comput. Appl.*, 82:93–113, 2017.
3. Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In *ITCS*, 2020.
4. Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *ICML*, 2020.
5. Antonios Antoniadis, Peyman Jabbarzade Ganje, and Golnoosh Shahkarami. A novel prediction setup for online speed-scaling. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2022, June 27-29, 2022, Tórshavn, Faroe Islands*, volume 227 of *LIPIcs*, pages 9:1–9:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
6. Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In *NeurIPS*, 2020.
7. Evripidis Bampis, Konstantinos Dogeas, Alexander V. Kononov, Giorgio Lucarelli, and Fanny Pascual. Scheduling with untrusted predictions. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 4581–4587. ijcai.org, 2022.
8. Evripidis Bampis, Bruno Escoffier, Themis Gouleakis, Niklas Hahn, Kostas Lakis, Golnoosh Shahkarami, and Michalis Xefteris. Learning-augmented online TSP on rings, trees, flowers and (almost) everywhere else. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.12.
9. Evripidis Bampis, Bruno Escoffier, and Michalis Xefteris. Canadian traveller problem with predictions. In Parinya Chalermsook and Bundit Laekhanukit, editors, *Approximation and Online Algorithms*, pages 116–133. Springer International Publishing, 2022.
10. Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
11. Bo Chen and Arjen P.A. Vestjens. Scheduling on identical machines: How good is lpt in an on-line setting? *Operations Research Letters*, 21(4):165–169, 1997.
12. Franziska Eberle, Alexander Lindermayr, Nicole Megow, Lukas Nölke, and Jens Schlöter. Robustification of online graph exploration methods. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):9732–9740, June 2022.
13. Donald K. Friesen. Tighter bounds for the multifit processor scheduling algorithm. *SIAM J. Comput.*, 13(1):170–181, 1984.
14. Donald K. Friesen and Michael A. Langston. Evaluation of a multifit-based scheduling algorithm. *J. Algorithms*, 7(1):35–59, 1986.
15. Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *ICML*, 2019.
16. Themis Gouleakis, Konstantinos Lakis, and Golnoosh Shahkarami. Learning-Augmented Algorithms for Online TSP on the Line. *To appear in AAAI*, 2023. CoRR abs/2206.00655.
17. Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical J.*, 45(9):1563–1581, 1966.
18. Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.
19. Dorit S. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing, 1997.
20. Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, 1988.

21 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *SPAA*, pages 285–294. ACM, 2021.

22 Edward G. Coffman Jr., M. R. Garey, and David S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput.*, 7(1):1–17, 1978.

23 Murali Kodialam and T. V. Lakshman. Prediction augmented segment routing. In *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6, 2021.

24 M.A. Langston. *A. Processors cheduling with improved heuristic algorithms. Doctoral dissertation,*. PhD thesis, Texas A&M Univ., College Station, Tex., 1981.

25 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *SODA*, pages 1859–1877. SIAM, 2020.

26 Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. 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 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

27 Joseph Leung, Laurie Kelly, and James H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., USA, 2004.

28 Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 357–368. ACM, 2022.

29 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311. PMLR, 2018.

30 Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

31 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In *Beyond Worst Case Analysis*, pages 646–662. Cambridge University Press, 2021.

32 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Communications of the ACM*, 65(7):33–35, 2022.

33 Narges Peyravi and Ali Moeini. Estimating runtime of a job in hadoop mapreduce. *Journal of Big Data*, 7(: 44), 2020.

34 Kirk Pruhs, Jirí Sgall, and Eric Torng. Online scheduling. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

35 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *NeurIPS*, 2018.

36 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *SODA*, 2020.

37 Sartaj Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 23(1):116–127, 1976.

38 Jirí Sgall. On-line scheduling. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, pages 196–231. Springer, 1998.

39 David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM J. Comput.*, 24(6):1313–1331, 1995.

40 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

41 Shufan Wang and Jian Li. Online algorithms for multi-shop ski rental with machine learned predictions. In *AAMAS*, 2020.

42 Alexander Wei. Better and simpler learning-augmented online caching. In *APPROX/RANDOM*, 2020.

43 Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *NeurIPS*, 2020.

44 Hirochika Yamashiro and Hirofumi Nonaka. Estimation of processing time using machine learning and real factory data for optimization of parallel machine scheduling problem. *Operations Research Perspectives*, 8(: 100196), 2021.

45 Tianming Zhao, Wei Li, and Albert Y. Zomaya. Uniform machine scheduling with predictions. In Akshat Kumar, Sylvie Thiébaux, Pradeep Varakantham, and William Yeoh, editors, *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, pages 413–422. AAAI Press, 2022.

Small-Space Algorithms for the Online Language Distance Problem for Palindromes and Squares

Gabriel Bathie
DIENS, École normale supérieure de Paris, PSL Research University, France
LaBRI, Université de Bordeaux, France

Tomasz Kociumaka
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Tatiana Starikovskaya
DIENS, École normale supérieure de Paris, PSL Research University, France

Abstract

We study the online variant of the language distance problem for two classical formal languages, the language of palindromes and the language of squares, and for the two most fundamental distances, the Hamming distance and the edit (Levenshtein) distance. In this problem, defined for a fixed formal language L, we are given a string T of length n, and the task is to compute the minimal distance to L from *every* prefix of T. We focus on the low-distance regime, where one must compute only the distances smaller than a given threshold k. In this work, our contribution is twofold:

1. First, we show *streaming* algorithms, which access the input string T only through a single left-to-right scan. Both for palindromes and squares, our algorithms use $O(k \operatorname{polylog} n)$ space and time per character in the Hamming-distance case and $O(k^2 \operatorname{polylog} n)$ space and time per character in the edit-distance case. These algorithms are randomised by necessity, and they err with probability inverse-polynomial in n.
2. Second, we show *deterministic read-only* online algorithms, which are also provided with read-only random access to the already processed characters of T. Both for palindromes and squares, our algorithms use $O(k \operatorname{polylog} n)$ space and time per character in the Hamming-distance case and $O(k^4 \operatorname{polylog} n)$ space and amortised time per character in the edit-distance case.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Pattern matching

Keywords and phrases Approximate pattern matching, streaming algorithms, palindromes, squares

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.10

Related Version *Full Version with Details for the Edit Distance Algorithms*: `https://arxiv.org/abs/2309.14788`

Funding Partially funded by the grant ANR-20-CE48-0001 from the French National Research Agency.

1 Introduction

The *language distance* problem is one of the most fundamental problems in formal language theory. In this problem, the task is to compute the minimal distance between a given string S and any string belonging to a formal language L. Introduced in the early 1970s by Aho and Peterson [2], the language distance problem has been studied extensively for regular languages under Hamming and edit distances [5], for general context-free languages, mainly focusing on the edit distance [1, 2, 8, 10, 25, 27, 29, 32, 33, 34], and the Dyck language (the language of well-nested parentheses sequences) in particular [1, 4, 8, 10, 12, 13, 15, 22, 23, 31, 32, 33].

Our results. In this work, we study the complexity of the online and low-distance version of the language distance problem, where we are given a string T of length n, and the task is to compute the minimal distance from *every* prefix of T to a formal language L (the distance and the language are specified in the problem definition). We focus on the low-distance regime, i.e., we assume to be given a threshold parameter k such that distances larger than k do not need to be computed. We consider the edit distance (defined as the minimum number of character insertions, deletions, and substitutions needed to transform one string into the other) and, as a preliminary step, the Hamming distance (allowing for substitutions only). We study the problem for two classical languages: the language PAL of all palindromes, where a palindrome is a string that is equal to its reversed copy, and the language SQ of all squares, where a square is the concatenation of two copies of a string. These two languages are very similar yet very different in nature: PAL is not regular but is context-free, whereas SQ is not even context-free. Formally, the problems we consider are defined as follows:

▶ **Problem 1.1.** k-LHD-PAL (resp. k-LHD-SQ)
Input: A string T of length n and a positive integer k.
Output: For each $1 \leq i \leq n$, report $\min\{k+1, hd_i\}$, where hd_i is the minimum Hamming distance between $T[1..i]$ and a string in PAL (resp. in SQ).

▶ **Problem 1.2.** k-LED-PAL (resp. k-LED-SQ)
Input: A string T of length n and a positive integer k.
Output: For each $1 \leq i \leq n$, report $\min\{k+1, ed_i\}$, where ed_i is the minimum edit distance between $T[1..i]$ and a string in PAL (resp. in SQ).

Table 1 Summary of the complexities of the algorithms introduced in this work.

Problem	Model	Time per character	Space complexity	Reference
k-LHD-PAL	Streaming	$O(k \log^3 n)$	$O(k \log n)$	Thm 3.2
k-LHD-SQ	Streaming	$\tilde{O}(k)$	$O(k \log^2 n)$	Thm 3.3
k-LHD-PAL/SQ	Read-only	$O(k \log n)$	$O(k \log n)$	Thms 4.8 and 4.10
k-LED-PAL/SQ	Streaming	$\tilde{O}(k^2)$	$\tilde{O}(k^2)$	Thms 5.1 and 5.2
k-LED-PAL/SQ	Read-only	$\tilde{O}(k^4)$ (amortised)	$\tilde{O}(k^4)$	Thms 5.3 and 5.4

Amir and Porat [3] showed that there is a randomised streaming algorithm that solves the k-LHD-PAL problem in $\tilde{O}(k)$ space and $\tilde{O}(k^2)$ time per input character.[1] We continue their line of research and show streaming algorithms for all four problems that use $\text{poly}(k, \log n)$ time per character and $\text{poly}(k, \log n)$ space. While streaming algorithms are extremely efficient (in particular, the space complexities above account for *all* the space used by the algorithms, including the space needed to store information about the input), it is important to note that they are randomised in nature, which means that they may produce incorrect results with a certain probability (inverse polynomial in the input size n). Motivated by this, we also study the problems in the read-only model, where random access to the input is allowed (and not accounted for in the space usage). In this model, we show *deterministic* algorithms for the four problems that use $\text{poly}(k, \log n)$ time per character and $\text{poly}(k, \log n)$ space; see Table 1 for a summary. As a side result of independent interest, we develop the first $\text{poly}(k, \log n)$ space read-only algorithms for computing k-mismatch and k-edit occurrences of a pattern in a text.

[1] Hereafter, $\tilde{O}(\cdot)$ hides factors polynomial in $\log n$.

Due to the lack of space, descriptions of the algorithms for the Language edit distance problems (k-LED-PAL and k-LED-SQ) are omitted from this version of the paper, but can be found in the full one.

1.1 Related work

Offline model. In the classical *offline* model, the problem of finding *all maximal substrings* that are within Hamming distance k from PAL can be solved in $O(nk)$ time as a simple application of the kangaroo jumps technique [17]. For the edit distance, Porto and Barbosa [28] showed an $O(nk^2)$ solution. For the SQ language, the best known solutions take $O(nk \log k +$ output) time for the Hamming distance [21] and $O(nk \log^2 k +$output) for the edit distance [24, 35, 36].

Online model. The problems k-LHD-PAL and k-LED-PAL can be viewed as a generalization of the classical online palindrome recognition problem (see [16] and references therein).

Streaming algorithms for PAL and SQ. Berebrink et al. [6] followed by Gawrychowski et al. [18] studied the question of computing the length of a maximal substring of a stream that belongs to PAL. Merkurev and Shur [26] considered a similar question for the SQ language.

2 Preliminaries

We assume to be given an alphabet Σ, the elements of which, called *characters*, can be stored in a single machine word of the RAM model. For an integer $n \geq 0$, we denote the set of all length-n strings by Σ^n, and we set $\Sigma^{\leq n} = \bigcup_{m=0}^{n} \Sigma^m$ as well as $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$. The empty string is denoted by ε.

For two strings $S, T \in \Sigma^*$, we use ST or $S \cdot T$ indifferently to denote their concatenation. For an integer $m \geq 0$, the string obtained by concatenating S to itself m times is denoted by S^m; note that $S^0 = \varepsilon$. A string S is a *square* if there exists a string T such that $S = T^2$.

For a string $T \in \Sigma^n$ and an index $i \in [1..n]$,[2] the ith character of T is denoted by $T[i]$. We use $|T| = n$ to denote the length of T. For indices $1 \leq i, j \leq n$, $T[i..j]$ denotes the substring $T[i]T[i+1]\cdots T[j]$ of T if $i \leq j$ and the empty string otherwise. When $i = 1$ or $j = n$, we omit these indices, i.e., we write $T[..j] = T[1..j]$ and $T[i..] = T[i..n]$. We extend the above notation with $T[i..j) = T[i..j-1]$ and $T(i..j] = T[i+1..j]$. We say that a string P is a *prefix* of T if there exists $j \in [1..n]$ such that $P = T[..j]$, and a *suffix* of T if there exists $i \in [1..n]$ such that $P = T[i..]$. We use T^R to denote the reverse of T, that is $T^R = T[n]T[n-1]\cdots T[1]$. A string T is a *palindrome* if $T^R = T$.

We define the *forward cyclic rotation* $\mathsf{rot}(T) = T[2]\cdots T[n]T[1]$. In general, a cyclic rotation $\mathsf{rot}^s(T)$ with shift $s \in \mathbb{Z}$ is obtained by iterating rot or the inverse operation rot^{-1}. A non-empty string $T \in \Sigma^n$ is *primitive* if it is distinct from its non-trivial rotations, i.e., if $T = \mathsf{rot}^s(T)$ holds only when n divides s.

Given two strings U, V and two indices $i \in [1..|U|]$ and $j \in [1..|V|]$, the *longest common prefix* (LCP) of $U[i..]$ and $V[j..]$, denoted $\mathsf{LCP}(U[i..], V[j..])$, is the length of the longest string that is a prefix of both $U[i..]$ and $V[j..]$.

[2] For integers $i, j \in \mathbb{Z}$, denote $[i..j] = \{k \in \mathbb{Z} : i \leq k \leq j\}$, $[i..j) = \{k \in \mathbb{Z} : i \leq k < j\}$, and $(i..j] = \{k \in \mathbb{Z} : i < k \leq j\}$.

Given two non-empty strings U, Q and an operator F defined over pairs of strings, we use the notation $F(U, Q^\infty)$ for the application of F to U and the prefix of $Q^\infty = QQ\cdots$ that has the same length as U, i.e., $F(U, Q^\infty) = F(U, Q^m[..|U|])$, where m is any integer such that $|Q^m| \geq |U|$. We define $F(Q^\infty, U)$ symmetrically.

2.1 Hamming distance, palindromes, and squares

The Hamming distance between two strings S, T (denoted $\mathsf{hd}(S, T)$) is defined to be equal to infinity if S and T have different lengths, and otherwise to the number of positions where the two strings differ (mismatches). We define the *mismatch information* between two length-n strings S and T, $\mathsf{MI}(S, T)$ as the set $\{(i, S[i], T[i]) : i \in [1..n] \text{ and } S[i] \neq T[i]\}$. For two strings P, T, a position $i \in [|P|..|T|]$ of T is a *k-mismatch occurrence* of P in T if $\mathsf{hd}(T(i-|P|..i], P) \leq k$. For an integer k, we denote $\mathsf{hd}_{\leq k}(X, Y) = \mathsf{hd}(X, Y)$ if $\mathsf{hd}(X, Y) \leq k$ and ∞ otherwise.

Due to the self-similarity of palindromes and squares, the Hamming distance from a string U to PAL and SQ can be measured in terms of the self-similarity of U.

▶ **Property 2.1.** *Each string $U \in \Sigma^m$ satisfies $\mathsf{hd}(U, PAL) = \mathsf{hd}(U[..\lfloor m/2 \rfloor], U(\lceil m/2 \rceil ..]^R) = \frac{1}{2}\mathsf{hd}(U, U^R)$.*

Proof. Denote $U_1 = U[..\lfloor m/2 \rfloor]$ and $U_2 = U(\lceil m/2 \rceil ..]$. For the second equality, we have $\mathsf{hd}(U, U^R) = \mathsf{hd}(U_1, U_2^R) + \mathsf{hd}(U_2, U_1^R) = 2 \cdot \mathsf{hd}(U_1, U_2^R)$.

The first equality is equivalent to $\mathsf{hd}(U_1, U_2^R) = \mathsf{hd}(U, \mathrm{PAL})$. As the Hamming distance between U and the palindrome $U_2^R U_2$ (or $U_2^R a U_2$ if m is odd) is $\mathsf{hd}(U_1, U_2^R)$, we have $\mathsf{hd}(U_1, U_2^R) \geq \mathsf{hd}(U, \mathrm{PAL})$.

Conversely, let V be a palindrome such that $\mathsf{hd}(U, V) = \mathsf{hd}(U, \mathrm{PAL})$. We decompose similarly V into $V_1 V_1^R$ (or $V_1 b V_1^R$ for odd m) and obtain $\mathsf{hd}(U, V) \geq \mathsf{hd}(U_1, V_1) + \mathsf{hd}(U_2, V_1^R)$. Using the fact that $\mathsf{hd}(U_2, V_1^R) = \mathsf{hd}(U_2^R, V_1)$ and applying the triangle inequality, we get $\mathsf{hd}(U_1, U_2^R) \leq \mathsf{hd}(U, \mathrm{PAL})$. ◀

▶ **Property 2.2.** *Each string $U \in \Sigma^m$ satisfies $\mathsf{hd}(U, SQ) = \mathsf{hd}(U[..m/2], U(m/2..])$ if m is even and $\mathsf{hd}(U, SQ) = \infty$ if m is odd.*

Proof. Every square has even length; hence, if m is odd, the distance between U and SQ is infinite. In what follows, we assume that $m = 2i$ for some $i \in \mathbb{N}$. Let $U_1 = U[..i]$ and $U_2 = U(i..]$. By modifying the copy of U_1 in U into U_2, we obtain a square $U_2 U_2$; hence, $\mathsf{hd}(U, \mathrm{SQ}) \leq \mathsf{hd}(U_1, U_2)$.

For the converse inequality, let V^2 be a square such that $\mathsf{hd}(U, \mathrm{SQ}) = \mathsf{hd}(U, V^2)$. We have $|V| = |U_1| = |U_2|$; hence, $\mathsf{hd}(U, V^2) = \mathsf{hd}(U_1, V) + \mathsf{hd}(V, U_2)$. Applying the triangle inequality, we obtain $\mathsf{hd}(U, \mathrm{SQ}) = \mathsf{hd}(U, V^2) \geq \mathsf{hd}(U_1, U_2)$. ◀

2.2 Models of computation

In this work, we focus on two by now classical models of computation: streaming and read-only random access. In the streaming model, we assume that the input string T arrives as a stream, one character at a time. For each prefix $T[1..i]$, we must report the distance to PAL or SQ as soon as we receive $T[i]$. We account for all the space used, including the space needed to store any information about T. In contrast, in the read-only model, we do not account for the space occupied by the input string. We assume that T is read from the left to the right. After having read $T[1..i]$, we assume to have constant-time read-only random access to the first i characters of T. Similar to the streaming model, the distance from $T[1..i]$ to PAL or SQ must be reported as soon as we read $T[i]$.

3 Warm-up: Streaming algorithms for the LHD problems

In this section, we present streaming algorithms for k-LHD-PAL and k-LHD-SQ. Our solutions use the Hamming distance sketches introduced by Clifford, Kociumaka, and Porat [11] to solve the streaming k-mismatch problem.

▶ **Fact 3.1.** *There exists a function* $\mathsf{sk}_k^{\mathsf{hd}}$ *(parameterized by a constant* $c > 1$*, integers* $n \geq k \geq 1$*, and a seed of* $O(\log n)$ *random bits) that assigns an* $O(k \log n)$*-bit sketch to each string in* $\Sigma^{\leq n}$*. Moreover:*

1. *There is an* $O(k \log^2 n)$*-time* encoding *algorithm that, given* $U \in \Sigma^{\leq k}$*, builds* $\mathsf{sk}_k^{\mathsf{hd}}(U)$*.*
2. *There is an* $O(k \log n)$*-time algorithm that, given any two among* $\mathsf{sk}_k^{\mathsf{hd}}(U), \mathsf{sk}_k^{\mathsf{hd}}(V)$*, or* $\mathsf{sk}_k^{\mathsf{hd}}(UV)$*, computes the third one (provided that* $|UV| \leq n$*).*
3. *There is an* $O(k \log^3 n)$*-time* decoding *algorithm that, given* $\mathsf{sk}_k^{\mathsf{hd}}(U)$ *and* $\mathsf{sk}_k^{\mathsf{hd}}(V)$*, computes* $\mathsf{MI}(U, V)$ *if* $\mathsf{hd}(U, V) \leq k$*. The error probability is* $O(n^{-c})$*.*

3.1 A streaming algorithm for k-LHD-PAL

We first show that the sketches described in Fact 3.1 give a simple algorithm improving upon the result of Amir and Porat [3] and achieving the time complexity of $\tilde{O}(k)$ per letter.

▶ **Theorem 3.2.** *There is a randomised streaming algorithm that solves the* k*-LHD-PAL problem for a string* $T \in \Sigma^n$ *using* $O(k \log n)$ *bits of space and* $O(k \log^3 n)$ *time per character. The algorithm errs with probability inverse-polynomial in* n*.*

Using Property 2.1, we can reduce the k-LHD-PAL problem to that of computing the threshold Hamming distance between the current prefix of the input string and its reverse. The algorithm maintains the sketches $\mathsf{sk}_{2k}^{\mathsf{hd}}(T[..i])$ and $\mathsf{sk}_{2k}^{\mathsf{hd}}(T[..i]^R)$. When it receives $T[i]$, it constructs $\mathsf{sk}_{2k}^{\mathsf{hd}}(T[i])$, updates both $\mathsf{sk}_{2k}^{\mathsf{hd}}(T[..i])$ and $\mathsf{sk}_{2k}^{\mathsf{hd}}(T[..i]^R)$, and computes $d = \mathsf{hd}_{\leq 2k}(T[..i], T[..i]^R)$ (in $O(k \log^3 n)$ total time by Fact 3.1). Property 2.1 implies $\mathsf{hd}_{\leq k}(T[..i], \mathrm{PAL}) = d/2$. The error probability of the algorithm follows from the error probability for the decoding algorithm for Hamming distance sketches.

The algorithm uses $O(k \log n)$ bits, which is nearly optimal: Indeed, by Property 2.1, if $U = VW$, with $|V| = |W|$, then $\mathsf{hd}(U, U^R) = 2 \cdot \mathsf{hd}(V, W^R)$. Therefore, using a standard reduction from streaming algorithms to one-way communication complexity protocols, we obtain a lower bound of $\Omega(k)$ bits for the space complexity of streaming algorithms for the k-LHD-PAL problem from the $\Omega(k)$ bits lower bound for the communication complexity of the Hamming distance [19].

3.2 A streaming algorithm for k-LHD-SQ

In this section, we show the following theorem:

▶ **Theorem 3.3.** *There is a randomised streaming algorithm that solves the* k*-LHD-SQ problem for a string* $T \in \Sigma^n$ *using* $O(k \log^2 n)$ *bits of space and* $\tilde{O}(k)$ *time per character. The algorithm errs with probability inverse-polynomial in* n*.*

Property 2.2 allows us to derive $\mathsf{hd}_{\leq k}(T[..2i], \mathrm{SQ})$ from the sketches $\mathsf{sk}_k^{\mathsf{hd}}(T[..i])$ and $\mathsf{sk}_k^{\mathsf{hd}}(T[..2i])$: we can combine them to obtain $\mathsf{sk}_k^{\mathsf{hd}}(T(i..2i])$, and a distance computation on $\mathsf{sk}_k^{\mathsf{hd}}(T[..i])$ and $\mathsf{sk}_k^{\mathsf{hd}}(T(i..2i])$ returns $\mathsf{hd}_{\leq k}(T[..i], T(i..2i]) = \mathsf{hd}_{\leq k}(T[..2i], \mathrm{SQ})$.

Naively applying this procedure requires storing the sketch $\mathsf{sk}_k^{\mathsf{hd}}(T[..i])$ until the algorithm has read $T[..2i]$, that is, storing $\Theta(n)$ sketches at the same time. To reduce the number of sketches stored, we use a filtering procedure based on the following observation:

▶ **Observation 3.4.** *If* $\mathsf{hd}(T[..2i], SQ) \leq k$ *and* $\ell \in [1..i]$*, then* $i + \ell$ *is a* k*-mismatch occurrence of* $T[..\ell]$*, that is,* $\mathsf{hd}(T[..\ell], T(i..i+\ell]) \leq k$*.*

▶ **Example 3.5.** For $k = 1, \ell = 2$, and $i = 3$, the word $T[..6] = \mathtt{abcacc}$ is a 1-mismatch square (by Property 2.2) and the fragment $T(3..5] = \mathtt{ac}$ is a 1-mismatch occurrence of the prefix $T[..2] = \mathtt{ab}$.

Observation 3.4 motivates our filtering procedure: if we choose some prefix $P = T[..\ell]$ of the string, we only need to store every $i \geq \ell$ such that $i + \ell$ is a k-mismatch occurrence of P. Clifford, Kociumaka and Porat [11] showed a data structure $\mathcal{S}$ that exploits the structure of such occurrences and stores them using $O(k \log^2 n)$ bits of space while allowing reporting the occurrence at position $i + \ell$ when $T[i + \ell + \Delta]$ is pushed into $\mathcal{S}$ – we say that $\mathcal{S}$ reports the k-mismatch occurrences of P in T with *a fixed delay* Δ [11]. Our algorithm needs to receive the occurrence at position $i + \ell$ when $T[2i]$ is pushed into the stream, i.e., we require $\mathcal{S}$ to report occurrences with *non-decreasing* delays. In Section 3.2.1 we present a modification of the data structure [11] to allow non-decreasing delays, and in Section 3.2.2 we explain how we use it to implement a space-efficient streaming algorithm for k-LHD-SQ.

3.2.1 Reporting k-mismatch occurrences with nondecreasing delay

The algorithm of Clifford, Kociumaka, and Porat [11] reports additional information along with the positions of the k-mismatch occurrences: specifically, it produces the *stream of k-mismatch occurrences of P in T*, defined as follows.

▶ **Definition 3.6** ([11, Definition 3.2])**.** *The stream of k-mismatch occurrences of a pattern P in a text T is a sequence S_P^k such that $S_P^k[i] = (i, \mathsf{MI}(T(i - |P|..i], P), \mathsf{sk}_k^{\mathsf{hd}}(T[..i - |P|]))$ if $\mathsf{hd}(P, T(i - |P|..i]) \leq k$ and $S_P^k[i] = \perp$ otherwise.*

As explained next, the algorithm of [11] can report the k-mismatch occurrences with a prescribed delay.

▶ **Corollary 3.7** (of [11])**.** *There is a streaming algorithm that, given a pattern P followed by a text $T \in \Sigma^n$, reports the k-mismatch occurrences of P in T using $O(k \log^2 n)$ bits of space and $O(\sqrt{k \log^3 n} + \log^4 n)$ time per character. The algorithm can report each occurrence i with no delay (that is, upon receiving $T[i]$) or with any prescribed delay $\Delta = \Theta(|P|)$ (that is, upon receiving $T[i + \Delta]$). For each reported occurrence i, the underlying tuple $S_P^k[i]$ can be provided on request in $O(k \log^2 n)$ time.*

Proof. If no delay is required, we use [11, Theorem 1.2], which reports k-mismatch occurrences of P in T and, upon request, provides the mismatch information $\mathsf{MI}(T(i - |P|..i], P)$; this algorithm uses $O(k \log^2 n)$ bits of space and takes $O(\sqrt{k \log^3 n} + \log^4 n)$ time per character. We also use [11, Fact 4.4] to maintain the sketch $\mathsf{sk}_k^{\mathsf{hd}}(T[..i])$ (reported on request); this algorithm uses $O(k \log n)$ bits of space and takes $O(\log^2 n)$ time per character.

Whenever requested to provide $S_P^k[i]$ for some k-mismatch occurrence i of P in T, we retrieve the mismatch information $\mathsf{MI}(T(i - |P|..i], P)$ (in $O(k)$ time) and the sketch $\mathsf{sk}_k^{\mathsf{hd}}(T[..i])$ (in $O(k \log^2 n)$ time). Combining $\mathsf{sk}_k^{\mathsf{hd}}(P)$ with $\mathsf{MI}(T(i - |P|..i], P)$, we build $\mathsf{sk}_k^{\mathsf{hd}}(T(i-|P|..i])$ (using [11, Lemma 6.4] in $O(k \log^2 n)$ time) and then derive $\mathsf{sk}_k^{\mathsf{hd}}(T[..i-|P|])$ using Fact 3.1 (in $O(k \log n)$ time). Overall, processing the request takes $O(k \log^2 n)$ time and $O(k \log^2 n)$ bits of space.

If a delay $\Delta = \Theta(|P|)$ is required, our approach depends on whether there exists $p \in [1..k]$ such that $\mathsf{hd}(P[..|P| - p], P(p..|P|]) \leq 2k$ (such p is called a $2k$-period in [11]). This property is tested using a streaming algorithm of [11, Lemma 4.3], which takes $O(k \log n)$

bits of space, $O(\sqrt{k \log n})$ time per character of P, and requires $O(k\sqrt{k \log n})$-time post-processing (performed while reading $T[..k]$). If P satisfies this condition, then we just use [11, Theorem 4.2], whose statement matches that of Corollary 3.7.

Otherwise, [11, Observation 4.1] shows that P has at most one k-mismatch occurrence among any k consecutive positions in T. In that case, we use the aforementioned approach to produce the stream S_P^k with no delay and the buffer of [11, Proposition 3.3] to delay the stream by Δ characters. The buffering algorithm takes $O(k \log^2 n)$ bits of space and processes each character $T[i]$ in $O(k \log^2 n + \log^3 n)$ time (if P has k-mismatch occurrences at positions i or $i - \Delta$) or $O(\sqrt{k \log n} + \log^3 n)$ time (otherwise). Since the former case holds for at most two out of every k consecutive positions, we can achieve $O(\sqrt{k \log^3 n} + \log^4 n)$ worst-case time per character by decreasing the delay to $\Delta - k$ and buffering up to k characters of T and up to k elements of S_P^k. While the algorithm processes $T[i + \Delta]$, the latter buffer already contains $S_P^k[i]$, but $O(k)$ time is still needed to output this value (if $S_P^k[i] \neq \bot$). ◀

The algorithm of Corollary 3.7 has a fixed delay Δ, i.e., it outputs $S_P^k[i]$ upon receiving $T[i + \Delta]$. Our application requires a variable delay: we need to access $S_P^k[i + |P|]$ upon reading $T[2i]$, that is, with a delay of $i - |P|$. We present a black-box construction that extends the data structure of Corollary 3.7 to support non-decreasing delays Δ_i, $i \in [1..d]$. Naively, one could use the algorithm $\mathcal{A}$ of Corollary 3.7 with a fixed delay Δ_1 and buffer the input characters so that $\mathcal{A}$ receives $T[i + \Delta_1]$ only when we actually process $T[i + \Delta_i]$. Unfortunately, this requires storing $T[i + \Delta_1..i + \Delta_i)$, which could take too much space. Thus, we feed $\mathcal{A}$ with $T[1..\Delta_1]$ followed by blank characters $\bot$ (issued at appropriate time steps without the necessity of buffering input characters) so that $\mathcal{A}$ reports k-mismatch occurrences $i \in [1..\Delta_1]$ with prescribed delays. Then, we use another instance of the algorithm of Corollary 3.7, with a fixed delay $\Delta_{1+\Delta_1}$, to output k-mismatch occurrences $i \in (\Delta_1..\Delta_1 + \Delta_{1+\Delta_1}]$; we continue this way until the whole interval $[1..d]$ is covered. We formalise this idea in the following lemma.

▶ **Lemma 3.8.** *Let $\Delta_1 \leq \Delta_2 \leq \cdots \leq \Delta_d$ be a non-decreasing sequence of $d = O(|P|)$ integers $\Delta_i = \Theta(|P|)$, represented by an oracle that reports each element Δ_i in constant time.*

There is a streaming algorithm that, given a pattern P followed by a text T, reports the k-mismatch occurrences of P in T using $O(k \log^2 n)$ bits of space and $O(\sqrt{k \log^3 n} + \log^4 n)$ time per character. The algorithm reports each occurrence $i \in [1..d]$ with delay Δ_i, that is, upon receiving $T[i + \Delta_i]$. For each reported occurrence $i \in [1..d]$, the underlying tuple $S_P^k[i]$ can be provided on request in $O(k \log n)$ time.

Proof. We use multiple instances $\mathcal{A}_1, \ldots, \mathcal{A}_t$ of the algorithm of Corollary 3.7. We define a sequence $(s_r)_{r=0}^t$ so that $\mathcal{A}_r$ works with a fixed delay $\Delta_{s_{r-1}}$, it is given $T[1..s_r) \cdot \bot^{\Delta_{s_{r-1}}}$, and it reports k-mismatch occurrences $i \in [s_{r-1}..s_r)$. Specifically, we set $s_0 = 1$ and $s_r = s_{r-1} + \Delta_{s_{r-1}}$, with t chosen as the smallest integer such that $s_t > d$. Note that $s_r - s_{r-1} = \Delta_{s_{r-1}} \geq \Delta_1$ implies $t \leq 1 + \frac{d}{\Delta_1} = O(1)$.

We assign three different roles to the algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_r$: *passive*, *active*, and *inactive*. While we process $T[j]$, the algorithm $\mathcal{A}_r$ is passive if $j < s_r$, active if $j \in [s_r..s_{r+1})$, and inactive if $j \geq s_{r+1}$. Our invariant is that, once we process $T[j]$, each passive algorithm $\mathcal{A}_r$ has already received $T[1..j]$, the unique active algorithm $\mathcal{A}_r$ has already received $T[1..s_r) \cdot \bot^{1+i-s_{r-1}}$, where i is the largest integer such that $i + \Delta_i \leq j$, and each inactive algorithm $\mathcal{A}_r$ has already received its entire input, that is, $T[1..s_r) \cdot \bot^{\Delta_{s_{r-1}}}$.

Upon receiving $T[j]$, we simply forward $T[j]$ to all passive algorithms. Moreover, if $j = i + \Delta_i$ for some $i \in [1..d]$, we feed the active algorithm with $\bot$ so that it checks whether i is a k-mismatch occurrence of P in T and, upon request, outputs $S_P^k[i]$.

Let us argue that this approach is correct from the perspective of a fixed algorithm $\mathcal{A}_r$. As we process $T[1..s_r)$, the algorithm is passive, and it is fed with subsequent characters of T. For $j = s_r - 1$, the position $i = s_{r-1} - 1$ is the maximum one such that $i + \Delta_i \le j$. Consequently, the input $T[1..s_r)$ already satisfies the invariant for passive algorithms. For subsequent iterations $j \in [s_r..s_{r+1})$, as $\mathcal{A}_r$ is active, it receives $\perp$ whenever i increases, so its input stays equal to $T[1..s_r) \cdot \perp^{1+i-s_{r-1}}$. The length of this string is $s_r + i - s_{r-1} = i + \Delta_{s_{r-1}}$, so the algorithm indeed checks whether i is a k-mismatch occurrence of P in T at each such iteration (recall that its fixed delay is $\Delta_{s_{r-1}}$), and it satisfies the invariant for active algorithms. Once we reach $j = s_{r+1} - 1$, we have $i = s_r - 1 = s_{r-1} + \Delta_{s_{r-1}} - 1$, so the input becomes $T[1..s_r) \cdot \perp^{\Delta_{s_{r-1}}}$, and it already satisfies the invariant for inactive algorithms. The state of inactive algorithms does not change, so this invariant remains satisfied as $\mathcal{A}_r$ stays inactive indefinitely.

The time and space complexity analysis follows from the fact that $t = O(1)$. ◀

3.2.2 Algorithm

We now show how to use the data structure of Lemma 3.8 to implement our filtering procedure using low space. For each $j \in [1..\lfloor \log n \rfloor]$, let P_j denote the prefix of the text of length $\ell_j = 2^j$, i.e., $P_j = T[..2^j]$. We search for k-mismatch occurrences of P_j in $T_j = T(3\ell_j/2..4\ell_j]$. As argued below, this allows filtering positions in $(3\ell_j..6\ell_j]$. Additionally, our choice of $(\ell_j)_j$ ensures that we do not miss any k-mismatch square when running our search for every P_j in parallel.

▷ Claim 3.9. For each $j \in [1..\lfloor \log n \rfloor]$, let Occ_j be the set of k-mismatch occurrences of P_j in $T_j = T(3\ell_j/2..4\ell_j]$. If $\mathsf{hd}(T[..2i], \mathrm{SQ}) \le k$ and $2i \in [3\ell_j..6\ell_j)$, then $p = i - \ell_j/2 \in \mathsf{Occ}_j$.

Proof. Since $\ell_j \le i$, Observation 3.4 implies that $i + \ell_j$ is a k-mismatch occurrence of P_j in T. Moreover, when $2i \in [3\ell_j..6\ell_j)$, we have $3\ell_j/2 \le i \le 3\ell_j$; therefore, that k-mismatch occurrence of P_j is fully contained within T_j, and it ends at positions $i + \ell_j - 3\ell_j/2 = i - \ell_j/2$ of T_j. ◁

In what follows, we use p to denote indices in T_j, whereas i denotes indices in the original text T. As $T_j = T(3\ell_j/2..4\ell_j]$, the correspondence is given by $i = p + 3\ell_j/2$. In other words, we only need to compute $\mathsf{hd}_{\le k}(T[..2i], \mathrm{SQ})$ when $i - \ell_j/2 \in \mathsf{Occ}_j$. As noted in Property 2.2, it suffices to know the sketches $\mathsf{sk}^{\mathsf{hd}}_k(T(i..2i])$ and $\mathsf{sk}^{\mathsf{hd}}_k(T[..i])$. We store $\mathsf{sk}^{\mathsf{hd}}_k(P_j) = \mathsf{sk}^{\mathsf{hd}}_k(T[..\ell_j])$ as well as $s_j = \mathsf{sk}^{\mathsf{hd}}_k(T[..3\ell_j/2])$ and maintain $\mathsf{sk}^{\mathsf{hd}}_k(T[..2i])$ in a rolling manner as we receive the characters of the text.

We use the algorithm of Lemma 3.8, asking for k-mismatch occurrences of P_j in T_j, to report $\mathsf{sk}^{\mathsf{hd}}_k(T_j[..i - \ell_j]) = \mathsf{sk}^{\mathsf{hd}}_k(T(\ell_j..i])$ for every $i \in \mathsf{Occ}_j$. The delay sequence is specified as $\Delta_p = p - \ell_j/2$ for $p \in [\ell_j..5\ell_j/2)$ so that the conditions of Lemma 3.8 are satisfied. (For $p < \ell_j$, we can assume $\Delta_p = \Delta_{\ell_j} = \ell_j/2$; anyway, there cannot be a k-mismatch occurrence of P_j before position ℓ_j.) This way, for every $i \in [3\ell_j/2..3\ell_j)$, we receive $S^k_{P_j}[i + \ell_j]$ (which corresponds to a potential k-mismatch occurrence starting at position $i + 1$) while processing $T_j[p + \Delta_p]$ for $p = i + \ell_j - 3\ell_j/2 = i - \ell_j/2$. As $\Delta_p = p - \ell_j/2$, this corresponds to position $p' = 2p - \ell_j/2$ in T_j, or position $i' = 2p + \ell_j = 2i$ in T, i.e., this happens precisely as we are processing $T[2i]$. See Figure 1 for an illustration of the above. If $S^k_{P_j}[i + \ell_j]$ is blank, we move on to the next position. Otherwise, we retrieve the sketch $\mathsf{sk}^{\mathsf{hd}}_k(T_j[..i]) = \mathsf{sk}^{\mathsf{hd}}_k(T(3\ell_j/2..i])$, combine it with $s_j = \mathsf{sk}^{\mathsf{hd}}_k(T[..3\ell_j/2])$ and $\mathsf{sk}^{\mathsf{hd}}_k(T[..2i])$ to obtain $\mathsf{sk}^{\mathsf{hd}}_k(T[..i])$ and $\mathsf{sk}^{\mathsf{hd}}_k(T(i..2i])$, and use the latter two sketches to compute $\mathsf{hd}_{\le k}(T[..i], T(i..2i])$, which is equal to $\mathsf{hd}_{\le k}(T[..2i], \mathrm{SQ})$ by Property 2.2.

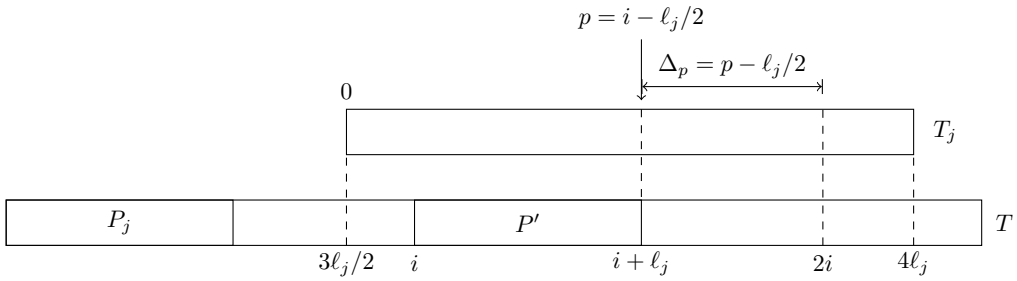


Figure 1 Illustration of our filtering procedure. Here, P' is a k-mismatch occurrence of P_j at position $i + \ell_j$ in T and position $p = i - \ell_j/2$ in T_j, reported with delay $\Delta_p = p - \ell_j/2$ in T_j, hence it arrives at time $2i$ in T.

We proceed with the complexity analysis of our algorithm. The k-mismatch pattern matching algorithm of Lemma 3.8 uses $O(k \log^2 n)$ bits of space and $\tilde{O}(k)$ time per character, and we maintain $O(\log n)$ instances of this algorithm. However, since all the patterns P_j are prefixes of T, the instances can share the pattern processing phase. Moreover, since any position is contained in at most three fragments $T[\ell_j..6\ell_j)$ (each such fragment follows P_j and contains T_j), at most three instances contribute to the time and space complexity at any given moment. Thus, the entire algorithm uses $O(k \log^2 n)$ bits of space and $\tilde{O}(k)$ time per character, which completes the proof of Theorem 3.3.

Our streaming algorithm for k-LED-SQ (Theorem 5.2) relies on the streaming algorithm for k-LHD-SQ. It requires testing $\mathsf{hd}(T[..2i], \mathrm{SQ}) \le k$ only for selected positions i, and thus it benefits from the following variant of Theorem 3.3:

▶ **Proposition 3.10.** *There is a randomised streaming algorithm that, given a string $T \in \Sigma^n$, upon receiving $T[2i]$, can be requested to test whether $\mathsf{hd}(T[..2i], SQ) \le k$ and, if so, report the mismatch information between $T[..2i]$ and a closest square. The algorithm uses $O(k \log^2 n)$ bits of space and processes each character in $\tilde{O}(\sqrt{k})$ or $\tilde{O}(k)$ time, depending on whether the request has been issued at that character.*

Proof. We follow the algorithm above with minor modifications. First, instead of maintaining $\mathsf{sk}^{\mathsf{hd}}_k(T[..2i])$ explicitly, we apply [11, Fact 4.4], which uses $O(k \log n)$ bits of space, takes $O(\log^2 n)$ time per character, and reports $\mathsf{sk}^{\mathsf{hd}}_k(T[..2i])$ on demand in $O(k \log^2 n)$ time.

To process a request concerning position $2i$, we retrieve $\mathsf{sk}^{\mathsf{hd}}_k(T[..2i])$ and ask the pattern-matching algorithm of Lemma 3.8 to output $S^k_{P_j}[i]$ (normally, the algorithm only reports whether i is a k-mismatch occurrence of P_j in T_j). In this case, we build $\mathsf{sk}^{\mathsf{hd}}_k(T[..i])$ and $\mathsf{sk}^{\mathsf{hd}}_k(T(i..2i])$ as in algorithm above. The decoding algorithm not only results in $\mathsf{hd}_{\le k}(T[..i], T(i..2i]) = \mathsf{hd}_{\le k}(T[..2i], \mathrm{SQ})$ but, if $\mathsf{hd}(T[..2i], \mathrm{SQ}) \le k$, also the underlying mismatch information.

The space complexity of the modified algorithm is still $O(k \log^2 n)$ bits. The running time is $\tilde{O}(\sqrt{k})$ if we do not ask the algorithm to test $\mathsf{hd}(T[..2i], \mathrm{SQ}) \le k$ and $\tilde{O}(k)$ if we do. ◀

4 Deterministic read-only algorithms for the LHD problems

In this section, we present deterministic read-only algorithms for k-LHD-PAL and k-LHD-SQ. We start by recalling structural results for k-mismatch occurrences used by the algorithms.

4.1 Structure of k-mismatch occurrences

▶ **Definition 4.1** ([9]). *A string U is d-mismatch periodic if there exists a primitive string Q such that $|Q| \leq |U|/128d$ and $\mathsf{hd}(U, Q^\infty) \leq 2d$. Such a string Q is called the* d-mismatch period *of U.*

The condition $|Q| \leq |U|/128d$ implies that Q is equal to some substring of U; hence, given the starting and ending positions of Q in U and random access to U, we can simulate random access to Q.

▷ Claim 4.2 (From [20, Claim 7.1]). Let U and V be strings such that U is a prefix of V, and $|V| \leq 2|U|$. If U is d-mismatch periodic with d-mismatch period Q, then V either is not d-mismatch periodic or has d-mismatch period Q.

Charalampopoulos, Kociumaka, and Wellnitz [9] showed that the set of k-mismatch occurrences has a very regular structure:

▶ **Fact 4.3** (See [9, Section 3]). *Let P and T be two strings such that $|P| \leq |T| \leq 3/2|P|$.*
1. *If P is not k-mismatch periodic, then there are $O(k)$ k-mismatch occurrences of P in T.*
2. *If P is k-mismatch periodic with period Q, then any two k-mismatch occurrences $i \leq i'$ of P in T satisfy $i \equiv i' \pmod{|Q|}$ and $\mathsf{hd}(T(i - |P|..i'], Q^\infty) \leq 3k$.*

They also presented efficient offline algorithms for computing the k-mismatch period and the k-mismatch occurrences in the so-called PILLAR model. In this model, one is given a family of strings $\mathcal{X}$ for preprocessing. The elementary objects are fragments $X[i..j]$ of strings $X \in \mathcal{X}$. Given elementary objects S, S_1, S_2, the PILLAR operations are:
1. $\mathsf{Access}(S, i)$: Assuming $i \in [1..|S|]$, retrieve $S[i]$.
2. $\mathsf{Length}(S)$: Retrieve the length $|S|$ of S.
3. $\mathsf{LCP}(S_1, S_2)$: Compute the length of the longest common prefix of S_1 and S_2.
4. $\mathsf{LCP}^R(S_1, S_2)$: Compute the length of the longest common suffix of S_1 and S_2.
5. $\mathsf{IPM}(S_1, S_2)$: Assuming that $|S_2| \leq 2|S_1|$, compute the set of the starting positions of occurrences of S_1 in S_2, which by Fine and Wilf periodicity lemma [14] can be represented as one arithmetic progression.

In the read-only model, operations Access and Length can be implemented in constant time and $O(\log m)$ bits. The operations LCP and LCP^R can be implemented naively via character-by-character comparison in $O(\min\{|S_1|, |S_2|\})$ total time and $O(\log m)$ bits. Finally, the IPM operation can be implemented in $O(|S_1| + |S_2|)$ total time and $O(\log m)$ bits (see e.g. [30]).

As a corollary, we immediately obtain:

▶ **Corollary 4.4** (From [9, Lemma 4.4]). *Given random access to a string U, testing whether it is d-mismatch periodic, and, if so, computing its d-mismatch period, can be done using $O(d|U|)$ time and $O(d)$ space.*

4.2 Read-only algorithm for the pattern matching with k mismatches

The above implementation of the PILLAR operations further implies an offline algorithm that finds all k-mismatch occurrences of P in T in $\tilde{O}(k^2 \cdot |T|)$ time and $\tilde{O}(k^2)$ space (see [9, Main Theorem 8]). Nevertheless, we provide a more efficient online algorithm that additionally provides the mismatch information for every k-mismatch occurrence of P.

▶ **Theorem 4.5.** *There is a deterministic online algorithm that finds all k-mismatch occurrences of a length-m pattern P within a text T using $O(k \log m)$ space and $O(k \log m)$ worst-case time per character. The algorithm outputs the mismatch information along with every reported k-mismatch occurrence of P.*

Consistently with the streaming algorithm of [11], our algorithm uses a family of exponentially-growing prefixes to filter out candidate positions. However, in order to use the structural properties of Fact 4.3 efficiently, we construct a different family $\mathcal{P}$ to ensure that we are either working in an approximately periodic region of the text or processing an aperiodic prefix.

We first add to $\mathcal{P}$ the prefixes $R_j = P[..\min\{m, \lfloor(3/2)^j\rfloor\}]$ for $j \in [0..\lceil\log_{3/2} m\rceil]$. If R_j is k-mismatch periodic but R_{j+1} is not, we also add to $\mathcal{P}$ the shortest extension of R_j that is not k-mismatch periodic. Hereafter, let $\mathcal{P} = (P_j)_{j=1}^{t}$ denote the resulting sequence of prefixes, sorted in order of increasing lengths, and let $\ell_j = |P_j|$ for every $j \in [1..t]$.

▷ Claim 4.6. The sequence $\mathcal{P} = (P_j)_{j=1}^{t}$ satisfies the following properties:

(a) $P_1 = P[1]$ and $P_t = P$,

(b) $t = |\mathcal{P}| = O(\log m)$,

(c) for every $j \in [1..t)$, we have $\ell_{j+1} \le 3\ell_j/2$,

(d) for every $j \in [1..t)$, if P_j is k-mismatch periodic with period Q_j, then $\mathsf{hd}(P_{j+1}, Q_j^\infty) \le 2k+1$.

Proof. Properties (a), (b), and (c) are straightforward. For Property (d), there are two possible cases: if P_{j+1} is k-mismatch periodic, Claim 4.2 implies that P_{j+1} has the same k-mismatch period Q_j as P_j, that is $\mathsf{hd}(P_{j+1}, Q_j^\infty) \le 2k$. Otherwise, by construction, P_{j+1} is the shortest extension of P_j that is not k-mismatch periodic. By minimality, removing its last character yields a k-mismatch periodic prefix, and by Claim 4.2, it has the same k-mismatch period Q_j as P_j, i.e., we have $\mathsf{hd}(P[..\ell_{j+1}), Q_j^\infty) \le 2k$ for $i < \ell_j$. Adding one more character to $P[..\ell_{j+1})$ can increase the Hamming distance by at most one. ◁

Processing the pattern. In the preprocessing phase, we build $\mathcal{P}$ and, for each k-mismatch periodic prefix $P_j \in \mathcal{P} \setminus \{P\}$, we also retrieve the period Q_j (represented as a fragment of P_j) and the mismatch information $\mathsf{MI}(P_{j+1}, Q_j^\infty)$. For subsequent indices $j \in [0..\lceil\log_{3/2} m\rceil]$, we add the prefix R_j to $\mathcal{P}$. If $R_j \neq P$, we apply Corollary 4.4 to test whether R_j is k-mismatch periodic and, if so, retrieve the period Q. If R_j is k-mismatch periodic, we build $\mathsf{MI}(R_j, Q^\infty)$ and extend R_j while maintaining the mismatch information with the appropriate prefix of Q^∞. We proceed until we reach length $|R_{j+1}|$ or $2k+1$ mismatches, whichever comes first. We add the obtained extension R'_j to $\mathcal{P}$ and store the mismatch information $\mathsf{MI}(R'_j, Q^\infty)$. If $\mathsf{hd}(R'_j, Q^\infty) \le 2k$, then $R'_j = R_{j+1}$ is k-mismatch periodic with the same period Q. Otherwise, by Claim 4.2, neither R'_j nor R_{j+1} are k-mismatch periodic. Processing each j takes $O(|R_{j+1}|k)$ time and $O(k)$ space, for a total of $O(mk)$ time and $O(k \log m)$ space across $j \in [0..\lceil\log_{3/2} m\rceil]$.

Processing the text. Our online algorithm processing the text T consists of $t = |\mathcal{P}|$ layers, each of which reports the k-mismatch occurrences of $P_j \in \mathcal{P}$, along with the underlying mismatch information.

The first layer, responsible for $P_1 = P[1]$, is implemented naively in $O(1)$ space and time per character.

Each of the subsequent layers receives the k-mismatch occurrences of P_j and outputs the k-mismatch occurrences of P_{j+1}. The processing is based on the following simple observation:

▶ **Observation 4.7.** *If P_{j+1} has a k-mismatch occurrence at position i of T, then P_j has a k-mismatch occurrence at position $i - \ell_{j+1} + \ell_j$ of T.*

We partition T into blocks of length $b := \lceil \ell_j/2 \rceil$ and, for each block $T(rb..(r+1)b]$, use a separate subroutine to output k-mismatch occurrences of P_{j+1} at positions $i \in (rb..(r+1)b]$. This subroutine receives the k-mismatch occurrences of P_j at positions $i - \ell_{j+1} + \ell_j \in (rb - \ell_{j+1} + \ell_j..(r+1)b - \ell_{j+1} + \ell_j]$. It is considered *active* as the algorithm reads $T(rb - \ell_{j+1} + \ell_j..(r+1)b]$; since $\ell_{j+1} \leq \frac{3}{2}\ell_j$, at most two subroutines are active at any given time. The implementation of the subroutine depends on whether P_j is k-mismatch periodic or not.

***P_j* is not *k*-mismatch periodic.** In this case, for every received k-mismatch occurrence i' of P_j, the subroutine stores the mismatch information $\mathsf{MI}(T(i' - \ell_j..i'], P_j)$ and, as the algorithm receives subsequent characters $T[i]$ for $i \in (i'..i'+\ell_{j+1}-\ell_j]$, we maintain $\mathsf{MI}(T(i'-\ell_j..i], P[..\ell_j + i - i'])$ as long as there are at most k mismatches. If this is still the case for $i = i' + \ell_{j+1} - \ell_j$, we report a k-mismatch occurrence of P_{j+1} and output $\mathsf{MI}(T(i' - \ell_j..i], P[..\ell_j + i - i']) = \mathsf{MI}(T(i-\ell_{j+1}..i], P_{j+1})$. By Observation 4.7, no k-mismatch occurrence of P_{j+1} is missed. Moreover, Fact 4.3 guarantees that the subroutine receives $O(k)$ k-mismatch occurrences of P_j, and thus it uses $O(k)$ space and $O(k)$ time per character.

***P_j* is *k*-mismatch periodic with period *Q_j*.** In this case, we wait for the leftmost k-mismatch occurrence $p \in (rb - \ell_{j+1} + \ell_j..(r+1)b - \ell_{j+1} + \ell_j]$ of P_j and ignore all the subsequent occurrences of P_j. We use the received mismatch information $\mathsf{MI}(T(p-\ell_j..p], P_j)$ and the preprocessed mismatch information $\mathsf{MI}(P_{j+1}, Q_j^\infty)$ to construct $\mathsf{MI}(T(p-\ell_j..p], Q_j^\infty)$; by the triangle inequality, the size of this set is guaranteed to be at most $3k$. As the algorithm receives subsequent characters of $T[i]$ for $i \in (p..(r+1)b]$, we maintain $\mathsf{MI}(T(p - \ell_j..i], Q_j^\infty)$ as long as the number of mismatches does not exceed $6k+1$. Whenever $i \geq p + \ell_{j+1} - \ell_j$ and $i \equiv p + \ell_{j+1} - \ell_j \pmod{|Q_j|}$, we extract $\mathsf{MI}(T(i-\ell_{j+1}..i], Q_j^\infty)$ from $\mathsf{MI}(T(p-\ell_j..i], Q_j^\infty)$ and use the precomputed mismatch information $\mathsf{MI}(P_{j+1}, Q_j^\infty)$ to construct $\mathsf{MI}(T(i-\ell_{j+1}..i], P_j)$. If it is of size at most k, we report i as a k-mismatch occurrence of P_j.

As for the correctness, we argue that we miss no k-mismatch occurrence $i \in (rb..(r+1)b]$ of P_{j+1} in T. Since $\mathsf{hd}(T(i - \ell_{j+1}..i], P_{j+1}) \leq k$ and $\mathsf{hd}(P_{j+1}, Q_j^\infty) \leq 2k+1$, we have $\mathsf{hd}(T(i - \ell_{j+1}..i], Q_j^\infty) \leq 3k+1$. Moreover, by Observation 4.7, $i - \ell_{j+1} + \ell_j$ is a k-mismatch occurrence of P_j. Fact 4.3 further implies that $i - \ell_{j+1} + \ell_j \equiv p \pmod{|Q_j|}$ and $\mathsf{hd}(T(p - \ell_j..i - \ell_{j+1}], Q_j^\infty) \leq 3k$. Consequently, $\mathsf{hd}(T(p - \ell_j..i], Q_j^\infty) \leq 6k+1$, and thus we compute $\mathsf{MI}(T(i - \ell_{j+1}..i], Q_j^\infty)$ and report i as a k-mismatch occurrence of P_{j+1}.

We conclude with the complexity analysis: the working space is $O(k)$, dominated by the maintained mismatch information. Moreover, whenever we compute $\mathsf{MI}(T(i - \ell_{j+1}..i], P_j)$, the size of this set is, by the triangle inequality, at most $6k+1+2k+1 \leq 8k+2$, and it can be computed in $O(k)$ time.

Summary. Overall, each subroutine of each level takes $O(k)$ space and $O(k)$ time per character. Since there are $t = O(\log m)$ levels and each level contains at most two active subroutines, the algorithm takes $O(k \log m)$ space and $O(k \log m)$ time per text character. Although our pattern preprocessing algorithm is an offline procedure, we can run it while the algorithm reads the first $m/2$ characters of the text. Then, while the algorithm reads further $m/2$ characters, it can process two characters at a time to catch up with the input stream. This does not result in any delay on the output because the leftmost k-mismatch occurrence of P is at position m or larger.

4.3 Read-only algorithm for k-LHD-PAL

▶ **Theorem 4.8.** *There is a deterministic online algorithm that solves the k-LHD-PAL problem for a string of length n using $O(k \log n)$ space and $O(k \log n)$ worst-case time per character.*

The algorithm uses a filtering approach to select positions where a prefix close to PAL can end. Define a family $\mathcal{P} = \{P_j = T[..\lfloor (3/2)^j \rfloor] : j \in [1..\lfloor \log_{3/2} n \rfloor]\}$ of prefixes of the text, and let $\ell_j = |P_j|$, setting $\ell_0 = 0$ for notational convenience.

▷ Claim 4.9. Consider $j \in [1..\lfloor \log_{3/2} n \rfloor]$ and a position $i \in (2\ell_{j-1}..2\ell_j]$. If $\mathsf{hd}(T[..i], \mathrm{PAL}) \le k$, then i is a $2k$-mismatch occurrence of P_j^R in T. Moreover, $\mathsf{hd}(T[..i], \mathrm{PAL}) = \mathsf{hd}(T(i-i'..i], P_j[1..i')^R)$ for $i' = \lfloor i/2 \rfloor$.

Proof. Note that $i > 2\ell_{j-1} \ge \ell_j$ implies that P_j is a prefix of $T[..i]$ and, equivalently, P_j^R is a suffix of $T[..i]^R$. Property 2.1 implies $2 \cdot \mathsf{hd}(T[..i], \mathrm{PAL}) = \mathsf{hd}(T[..i], T[..i]^R) \ge \mathsf{hd}(T(i-\ell_j..i], P_j)$. Thus, if $\mathsf{hd}(T[..i], \mathrm{PAL}) \le k$, then i is a $2k$-mismatch occurrence of P_j in T. Since $T[..i']$ is a prefix of P_j, Property 2.1 further implies $\mathsf{hd}(T[..i], \mathrm{PAL}) = \mathsf{hd}(T(i-i'..i], T[..i']^R) = \mathsf{hd}(T(i-i'..i], P_j[1..i')^R)$. ◁

The algorithm constructs the family $\mathcal{P}$ as it reads the text. For each level j, we implement a subroutine responsible for positions $i \in (2\ell_{j-1}..2\ell_j]$. First, while reading $T[\ell_j..2\ell_{j-1})$, we launch the pattern-matching algorithm of Theorem 4.5 in order to compute the $2k$-mismatch occurrences of P_j^R in $T_j = T[..2\ell_j)$ and feed the pattern-matching algorithm with the pattern P_j and a prefix $T[..2\ell_{j-1})$ of T_j, ignoring any output produced. The total number of characters provided is $\ell_j + 2\ell_{j-1} \le 7 \cdot (2\ell_{j-1} - \ell_j)$, so we can feed the algorithm with $O(1)$ characters for every scanned character of T. Then, while reading $T[2\ell_{j-1}..2\ell_j)$, we feed the pattern-matching algorithm with subsequent characters of T. For every reported $2k$-mismatch occurrence i of P_j^R in T_j, we retrieve the mismatch information $\mathsf{MI}(T(i-\ell_j..i], P_j^R)$ and obtain $\mathsf{MI}(T(i-i'..i], P_j[..i']^R)$ by removing the entries corresponding to the leftmost $\ell_j - i'$ positions. We report the size of this set (or ∞ if the size exceeds k) as $\mathsf{hd}_{\le k}(T[..i], \mathrm{PAL})$.

By Claim 4.9, all positions $i \in (2\ell_{j-1}..2\ell_j]$ such that $\mathsf{hd}(T[..i], \mathrm{PAL}) \le k$ pass the test and the distance $\mathsf{hd}(T[..i], \mathrm{PAL})$ is equal to the size of the set $\mathsf{MI}(T(i-i'..i], P_j[..i']^R)$. As for the complexity analysis, observe that, for each level j, the pattern-matching algorithm uses $O(k \cdot j)$ space and takes $O(k \cdot j)$ time per character. Since, at any time, there is a constant number of active levels, the main algorithm uses $O(k \log n)$ space and takes $O(k \log n)$ time per character.

4.4 Read-only algorithm for k-LHD-SQ

▶ **Theorem 4.10.** *There is a deterministic online algorithm that solves the k-LHD-SQ problem for a string $T \in \Sigma^n$ using $O(k \log n)$ space and $O(k \log n)$ worst-case time per character.*

Our algorithm is very similar to the pattern-matching algorithm of Theorem 4.5. We use the same sequence $\mathcal{P} = (P_j)_{j=1}^t$ of prefixes, now defined for $P = T$. Again, we set $\ell_j = |P_j|$ for $j \in [1..t]$. Instead of Observation 4.7, we use Observation 3.4 to argue that our filtering procedure is correct.

Processing $\mathcal{P}$. We build $\mathcal{P}$ in an online fashion so that the prefix P_j is constructed while scanning $T(\ell_j..\lceil 3\ell_j/2 \rceil]$. If P_j is k-mismatch periodic, then we also identify P_{j+1} and build $\mathsf{MI}(P_{j+1}, Q_j^\infty)$.

For subsequent indices $j \in [0..\lfloor \log_{3/2} n \rfloor]$, we add the prefix R_j to $\mathcal{P}$ as soon as it has been read. Then, we launch an offline procedure that applies Corollary 4.4 to test whether R_j is k-mismatch periodic and, if so, retrieves the period Q. If R_j is k-mismatch periodic, we build $\mathsf{MI}(R_j, Q^\infty)$ and extend R_j while maintaining the mismatch information with the appropriate prefix of Q^∞. We proceed until we reach length $|R_{j+1}|$ or $2k+1$ mismatches, whichever comes first. We add the obtained extension R'_j to $\mathcal{P}$ and store the mismatch information $\mathsf{MI}(R'_j, Q^\infty)$. If $\mathsf{hd}(R'_j, Q^\infty) \leq 2k$, then $R'_j = R_{j+1}$ is k-mismatch periodic with the same period Q. Otherwise, by Claim 4.2, neither R'_j nor R_{j+1} are k-mismatch periodic. Processing each j takes $O(|R_{j+1}|k)$ time and $O(k)$ space, and this computation needs to be completed while the algorithm reads $T(|R_j|..|R_{j+1}|]$. This gives $O(k)$ time per position since $\lfloor \frac{3}{2}|R_j| \rfloor \leq |R_{j+1}| \leq \lceil \frac{3}{2}|R_j| \rceil$.

Across all indices $j \in [0..\lfloor \log_{3/2} n \rfloor]$, the preprocessing algorithm takes $O(k)$ space and time per character (since no two indices are processed simultaneously).

Computing the distances. For each level $j \in [1..t]$, we implement a subroutine responsible for even positions $i \in [2\ell_j..2\ell_{j+1})$; this procedure is active as we read $T[\ell_j..2\ell_{j+1})$. As described above, the pattern P_j is identified while the algorithm reads $T(\ell_j..\lceil 3\ell_j/2 \rceil]$ and, if P_j is k-mismatch periodic, the period Q_j and the mismatch information $\mathsf{MI}(P_{j+1}, Q_j^\infty)$ are also computed at that time. While reading $T[\lceil 3\ell_j/2 \rceil..2\ell_j)$, we launch the pattern-matching algorithm of Theorem 4.5 to report the k-mismatch occurrences of P_j in $T_j = T[..\ell_j + \ell_{j+1})$ and feed this algorithm with the pattern P_j and the prefix $T[..2\ell_j)$ of the text T_j. The total number of characters provided is $3\ell_j \leq 6 \cdot \frac{1}{2}\ell_j$, so can feed the pattern-matching algorithm with $O(1)$ character for every scanned character of T. Then, while reading $T[2\ell_j..\ell_j + \ell_{j+1})$, we feed the pattern-matching algorithm subsequent text characters. For every $i' \in [2\ell_j..\ell_j + \ell_{j+1})$, we learn whether i' is a k-mismatch occurrence of P_j and, if so, we obtain the mismatch information $\mathsf{MI}(P_j, T(i' - \ell_j..i'])$. How we utilise this output depends on whether P_j is k-mismatch periodic or not: if P_j is not k-mismatch periodic, then T_j contains $O(k)$ k-mismatch occurrences of P_j and storing them explicitly requires little space. When P_j is k-mismatch periodic, T_j must exhibit similar periodicity, which we can use to avoid storing all occurrences explicitly.

***P_j* is not *k*-mismatch periodic.** In this case, for every received k-mismatch occurrence i' of P_j, we store the mismatch information $\mathsf{MI}(T(i' - \ell_j..i'], P_j)$ and, as the algorithm receives subsequent characters $T[i]$ for $i \in (i'..2(i' - \ell_j)]$, we maintain $\mathsf{MI}(T(i' - \ell_j..i], T[..\ell_j + i - i'])$ as long as there are at most k mismatches. If this is still the case for $i = 2(i' - \ell_j)$, we report that $T[..i]$ is a k-mismatch square, with $\mathsf{hd}(T[..i], \mathrm{SQ}) = \mathsf{hd}(T(i' - \ell_j..i], T[..\ell_j + i - i']) = \mathsf{hd}(T(i/2..i], T[..i/2])$. By Observation 3.4, no k-mismatch square $T[..i]$ is missed. Moreover, Fact 4.3 guarantees that there are $O(k)$ k-mismatch occurrences of P_j, and thus we use $O(k)$ space and $O(k)$ time per character to process all of them.

***P_j* is *k*-mismatch periodic with period *Q_j*.** In this case, we wait for the leftmost k-mismatch occurrence $p \in [2\ell_j..\ell_j + \ell_{j+1})$ of P_j and ignore all the subsequent occurrences of P_j. We use the received mismatch information $\mathsf{MI}(T(p - \ell_j..p], P_j)$ and the preprocessed mismatch information $\mathsf{MI}(P_{j+1}, Q_j^\infty)$ to construct $\mathsf{MI}(T(p - \ell_j..p], Q_j^\infty)$; by the triangle inequality, the size of this set is guaranteed to be at most $3k$. As the algorithm receives subsequent characters of $T[i]$ for $i \in (p..2\ell_{j+1})$, we maintain $\mathsf{MI}(T(p - \ell_j..i], Q_j^\infty)$ as long as the number of mismatches does not exceed $6k+1$. Whenever $i/2 \geq p - \ell_j$ and $i/2 \equiv p - \ell_j \pmod{|Q_j|}$, we extract $\mathsf{MI}(T(i/2..i], Q_j^\infty)$ from $\mathsf{MI}(T(p - \ell_j..i], Q_j^\infty)$ and use the precomputed

mismatch information $\mathsf{MI}(P_{j+1}, Q_j^\infty)$ to construct $\mathsf{MI}(T[..i/2], Q_j^\infty)$ first, and then derive $\mathsf{MI}(T[..i/2], T(i/2..i])$. If the latter is of size at most k, we report $T[..i]$ as a k-mismatch square.

As for the correctness, we argue that we miss no k-mismatch square $T[..i]$ with $i \in (2\ell_j..2\ell_{j+1}]$. Since $\mathsf{hd}(T(i/2..i], T[..i/2]) \le k$ and $\mathsf{hd}(P_{j+1}, Q_j^\infty) \le 2k+1$, as a corollary we obtain $\mathsf{hd}(T(i/2..i], Q_j^\infty) \le 3k+1$. Moreover, by Observation 3.4, $i/2 + \ell_j$ is a k-mismatch occurrence of P_j. Fact 4.3 further implies that $i/2 + \ell_j \equiv p \pmod{|Q_j|}$ and $\mathsf{hd}(T(p-\ell_j..i/2], Q_j^\infty) \le 3k$. Consequently, $\mathsf{hd}(T(p-\ell_j..i], Q_j^\infty) \le 6k+1$, and thus we compute $\mathsf{MI}(T[..i/2], T(i/2..i])$ and report $T[1..i]$ as a k-mismatch square.

We conclude with the complexity analysis: the working space is $O(k)$, dominated by the maintained mismatch information. Moreover, whenever we compute $\mathsf{MI}(T[..i/2], T(i/2..i])$, the size of this set is, by the triangle inequality, at most $6k+1+2k+1 \le 8k+2$, and it can be computed in $O(k)$ time.

Summary. Overall, each level takes $O(k \log n)$ space and $O(k \log n)$ time per character, dominated by the pattern-matching algorithm of Theorem 4.5. However, since constantly many levels are processed at any given time, the entire algorithm still uses $O(k \log n)$ space and $O(k \log n)$ time per character.

5 Language Edit Distance problems

The *edit distance* between two strings U and V, denoted by $\mathsf{ed}(U, V)$, is the minimum number of character insertions, deletions, and substitutions required to transform U into V. Similar to the Hamming distance, the edit distance from a string U to PAL and SQ can be expressed in terms of self-similarity of U. This allows us to use similar approaches as for the Language Hamming distance problems, with tools for the Hamming distance replaced with appropriate tools for the edit distance. Details for the proof of these theorems can be found in the full version of the paper, available on arXiv at `https://arxiv.org/abs/2309.14788`.

By replacing the Hamming distance sketch [11] with the edit distance sketch of Bhattacharya and Koucký [7].

▶ **Theorem 5.1.** *There is a randomised streaming algorithm that solves the k-LED-PAL problem for a string of length n using $\tilde{O}(k^2)$ bits of space and $\tilde{O}(k^2)$ time per character.*

Furthermore, the results of Bhattacharya and Koucký [7] show a reduction from the edit distance to the Hamming distance via locally consistent string decompositions, which allows reducing the k-LED-SQ problem to k-LHD-SQ, solved via Proposition 3.10:

▶ **Theorem 5.2.** *There is a randomised streaming algorithm that solves the k-LED-SQ problem for a string of length n using $\tilde{O}(k^2)$ bits of space and $\tilde{O}(k^2)$ time per character.*

Finally, by replacing the online read-only algorithm for finding the k-mismatch occurrences of a pattern in a text with an online read-only algorithm for finding k-error occurrences and the structural results for the Hamming distance with the structural results for the edit distance, we obtain algorithms for k-LED-PAL and k-LED-SQ:

▶ **Theorem 5.3.** *There is a deterministic online read-only algorithm that solves the k-LED-PAL problem for a string of length n using $\tilde{O}(k^4)$ bits of space and $\tilde{O}(k^4)$ time per character.*

▶ **Theorem 5.4.** *There is a deterministic online read-only algorithm that solves the k-LED-SQ problem for a string of length n using $\tilde{O}(k^4)$ bits of space and $\tilde{O}(k^4)$ amortised time per character.*

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the Current Clique Algorithms Are Optimal, so Is Valiant's Parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.

2 Alfred V. Aho and Thomas G. Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM Journal on Computing*, 1(4):305–312, 1972. doi:10.1137/0201022.

3 Amihood Amir and Benny Porat. Approximate on-line palindrome recognition, and applications. In *Proc. of CPM 2014*, volume 8486 of *LNCS*, pages 21–29. Springer, 2014. doi:10.1007/978-3-319-07566-2_3.

4 Arturs Backurs and Krzysztof Onak. Fast algorithms for parsing sequences of parentheses with few errors. In *Proc. of PODS 2016*, pages 477–488. ACM, 2016. doi:10.1145/2902251.2902304.

5 Djamal Belazzougui and Mathieu Raffinot. Approximate regular expression matching with multi-strings. *Journal of Discrete Algorithms*, 18:14–21, 2013. doi:10.1016/j.jda.2012.07.008.

6 Petra Berenbrink, Funda Ergün, Frederik Mallmann-Trenn, and Erfan Sadeqi Azer. Palindrome recognition in the streaming model. In *Proc. of STACS*, volume 25, pages 149–161, 2014. doi:10.4230/LIPIcs.STACS.2014.149.

7 Sudatta Bhattacharya and Michal Koucký. Locally consistent decomposition of strings with applications to edit distance sketching. In *Proc. of 55th STOC*, pages 219–232. ACM, 2023. doi:10.1145/3564246.3585239.

8 Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM Journal on Computing*, 48(2):481–512, 2019. doi:10.1137/17M112720X.

9 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *Proc. of 61st FOCS*, pages 978–989. IEEE, 2020. doi:10.1109/FOCS46700.2020.00095.

10 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *Proc. of 54th STOC*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057.

11 Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. In *Proc. of SODA 2019*, pages 1106–1125. SIAM, 2019. doi:10.1137/1.9781611975482.68.

12 Debarati Das, Tomasz Kociumaka, and Barna Saha. Improved approximation algorithms for Dyck edit distance and RNA folding. In *Proc. of ICALP 2022*, volume 229 of *LIPIcs*, pages 49:1–49:20, 2022. doi:10.4230/LIPIcs.ICALP.2022.49.

13 Anita Dürr. Improved bounds for rectangular monotone Min-Plus Product and applications. *Information Processing Letters*, 181:106358, 2023. doi:10.1016/j.ipl.2023.106358.

14 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.1090/S0002-9939-1965-0174934-9.

15 Dvir Fried, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, Ely Porat, and Tatiana Starikovskaya. An improved algorithm for the k-Dyck edit distance problem. In *Proc. of SODA 2022*, pages 3650–3669. SIAM, 2022. doi:10.1137/1.9781611977073.144.

16 Zvi Galil. Real-time algorithms for string-matching and palindrome recognition. In *Proc. of STOC*, pages 161–173. ACM, 1976. doi:10.1145/800113.803644.

17 Zvi Galil and Raffaele Giancarlo. Improved string matching with k mismatches. *ACM SIGACT News*, 17(4):52–54, 1986. doi:10.1145/8307.8309.

18 Pawel Gawrychowski, Oleg Merkurev, Arseny M. Shur, and Przemyslaw Uznanski. Tight tradeoffs for real-time approximation of longest palindromes in streams. *Algorithmica*, 81(9):3630–3654, 2019. doi:10.1007/s00453-019-00591-8.

19 Wei Huang, Yaoyun Shi, Shengyu Zhang, and Yufan Zhu. The communication complexity of the Hamming distance problem. *Information Processing Letters*, 99(4):149–153, 2006.

20 Tomasz Kociumaka, Ely Porat, and Tatiana Starikovskaya. Small-space and streaming pattern matching with k edits. In *Proc. of FOCS 2021*, pages 885–896. IEEE, 2021. doi:10.1109/FOCS52979.2021.00090.

21 Roman Kolpakov and Gregory Kucherov. Finding approximate repetitions under Hamming distance. *Theoretical Computer Science*, 303(1):135–156, 2003. Logic and Complexity in Computer Science. doi:10.1016/S0304-3975(02)00448-6.

22 Michal Koucký and Michael E. Saks. Simple, deterministic, fast (but weak) approximations to edit distance and Dyck edit distance. In *Proc. of SODA 2023*, pages 5203–5219. SIAM, 2023. doi:10.1137/1.9781611977554.ch188.

23 Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming algorithms for recognizing nearly well-parenthesized expressions. In *Proc. of MFCS 2011*, volume 6907 of *LNCS*, pages 412–423. Springer, 2011. doi:10.1007/978-3-642-22993-0_38.

24 Gad M. Landau and Jeanette P. Schmidt. An algorithm for approximate tandem repeats. In *Proc. of CPM*, pages 120–133, 1993. doi:10.1007/BFb0029801.

25 Lillian Lee. Fast context-free grammar parsing requires fast Boolean matrix multiplication. *Journal of the ACM*, 49(1):1–15, January 2002. doi:10.1145/505241.505242.

26 Oleg Merkurev and Arseny M. Shur. Computing the maximum exponent in a stream. *Algorithmica*, 84(3):742–756, 2022. doi:10.1007/s00453-021-00883-y.

27 Gene Myers. Approximately matching context-free languages. *Information Processing Letters*, 54(2):85–92, 1995. doi:10.1016/0020-0190(95)00007-y.

28 Alexandre H. L. Porto and Valmir Carneiro Barbosa. Finding approximate palindromes in strings. *Pattern Recognit.*, 35(11):2581–2591, 2002. doi:10.1016/S0031-3203(01)00179-0.

29 Walter L. Ruzzo. On the complexity of general context-free language parsing and recognition. In *Proc. of ICALP 1979*, volume 71 of *LNCS*, pages 489–497. Springer, 1979. doi:10.1007/3-540-09510-1_39.

30 Wojciech Rytter. On maximal suffixes and constant-space linear-time versions of KMP algorithm. *Theoretical Computer Science*, 299(1-3):763–774, 2003. doi:10.1016/S0304-3975(02)00590-X.

31 Barna Saha. The Dyck language edit distance problem in near-linear time. In *Proc. of FOCS 2014*, pages 611–620. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.71.

32 Barna Saha. Language edit distance and maximum likelihood parsing of stochastic grammars: Faster algorithms and connection to fundamental graph problems. In *Proc. of FOCS 2015*, pages 118–135. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.17.

33 Barna Saha. Fast space-efficient approximations of language edit distance and RNA folding: An amnesic dynamic programming approach. In *Proc. of FOCS 2017*, pages 295–306. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.35.

34 Giorgio Satta. Tree-adjoining grammar parsing and boolean matrix multiplication. *Comput. Linguistics*, 20(2):173–191, 1994. URL: https://aclanthology.org/J94-2002.

35 Dina Sokol, Gary Benson, and Justin Tojeira. Tandem repeats over the edit distance. *Bioinformatics*, 23(2):e30–e35, January 2007. doi:10.1093/bioinformatics/btl309.

36 Dina Sokol and Justin Tojeira. Speeding up the detection of tandem repeats over the edit distance. *Theoretical Computer Science*, 525:103–110, 2014. Advances in Stringology. doi:10.1016/j.tcs.2013.04.021.

Sparse Graphs of Twin-Width 2 Have Bounded Tree-Width

Benjamin Bergougnoux
University of Warsaw, Poland

Jakub Gajarský
University of Warsaw, Poland

Grzegorz Guśpiel
Masaryk University, Brno, Czech Republic

Petr Hliněný
Masaryk University, Brno, Czech Republic

Filip Pokrývka
Masaryk University, Brno, Czech Republic

Marek Sokołowski
University of Warsaw, Poland

Abstract

Twin-width is a structural width parameter introduced by Bonnet, Kim, Thomassé and Watrigant [FOCS 2020]. Very briefly, its essence is a gradual reduction (a contraction sequence) of the given graph down to a single vertex while maintaining limited difference of neighbourhoods of the vertices, and it can be seen as widely generalizing several other traditional structural parameters. Having such a sequence at hand allows to solve many otherwise hard problems efficiently. Our paper focuses on a comparison of twin-width to the more traditional tree-width on sparse graphs. Namely, we prove that if a graph G of twin-width at most 2 contains no $K_{t,t}$ subgraph for some integer t, then the tree-width of G is bounded by a polynomial function of t. As a consequence, for any sparse graph class $\mathcal{C}$ we obtain a polynomial time algorithm which for any input graph $G \in \mathcal{C}$ either outputs a contraction sequence of width at most c (where c depends only on $\mathcal{C}$), or correctly outputs that G has twin-width more than 2. On the other hand, we present an easy example of a graph class of twin-width 3 with unbounded tree-width, showing that our result cannot be extended to higher values of twin-width.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Fixed parameter tractability

Keywords and phrases twin-width, tree-width, excluded grid, sparsity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.11

Funding J. Gajarský and M. Sokołowski have received funding from the European Research Council (ERC) (grant agreement No 948057 – BOBR).

1 Introduction

Twin-width is a relatively new structural width measure of graphs and relational structures introduced in 2020 by Bonnet, Kim, Thomassé and Watrigant [13]. Informally, the twin-width of a graph measures how diverse the neighbourhoods of the graph vertices are. For instance, *cographs* – the graphs which can be built from singleton vertices by repeated operations of a disjoint union and taking the complement – are exactly the graphs of twin-width 0, which means that the graph can be brought down to a single vertex by successively identifying twin vertices. (Two vertices x and y are called *twins* in a graph G if they have the same neighbours in $V(G) \setminus \{x, y\}$.) Hence the name, *twin-width*, for the parameter.

Importance of this new concept is clearly witnessed by numerous recent papers on the topic, such as the follow-up series [7–11, 14] and more related research papers which consider twin-width from algorithmic [5, 25], combinatorial [2, 3, 23], structural [4, 15] and logical [4, 19] perspective.

In particular, twin-width, as a structural width parameter, has naturally algorithmic applications in the parameterized complexity area. Among the most important ones we mention that the first order (FO) model checking problem – that is, deciding whether a fixed first-order sentence holds in an input graph – can be solved in linear FPT-time [14]. This and other algorithmic applications assume that a contraction sequence of bounded width is given alongside with the input graph, since in general we do not know how to construct such a sequence efficiently. Consequently, finding an efficient algorithm for computing twin-width of input graph G together with its twin-width decomposition (known as a *contraction sequence* of G) is a central problem in the area. Currently, very little is known about this problem. It is known that one can check whether a graph has twin-width 0 and 1 and compute the corresponding contraction-sequence in polynomial time [12]. On the other hand, in general the problem of deciding the exact value of twin-width is NP-hard, and in particular, deciding whether a graph has twin-width 4 is NP-hard [5]. This means that even for fixed k, the best one can hope for is an approximation algorithm which for given input graph G either correctly outputs that G has twin-width more than k or produces a contraction sequence of G of width at most k', where k' is some fixed number with $k < k'$. However, to the best of our knowledge, no such efficient algorithm is currently known, even for graph classes of twin-width 2.

The importance and popularity of twin-width largely stems from the fact that it generalizes many well-known graph-theoretic concepts. For example, graph classes of bounded clique-width, planar graphs and more generally graph classes defined by excluding a fixed minor, posets of bounded width and various subclasses of geometric graphs have been shown to have bounded twin-width [14, 22]. In many cases, the proof of having bounded twin-width is constructive, meaning that for such class there exists an integer d and a polynomial time algorithm which takes a graph G as input and outputs a contraction sequence of G of width at most d.

Apart from establishing that some well-known graph classes have bounded twin-width, there are also results relating twin-width to various graph classes from the other direction: one considers graph classes of bounded bounded twin-width which are restricted in some sense, and proves that such classes fall within some well-studied framework. Prime examples of this approach are results stating that $K_{t,t}$-free graph classes of bounded twin-width have bounded expansion [8, 18] or that stable graph classes of bounded twin-width have structurally bounded expansion [20]. Results of this form allow us to use well-established structural and algorithmic tools for (structurally) bounded expansion to restricted classes of bounded twin-width.

Our research is motivated by the following conjecture, which also fits into this line of research and about which we learned from É. Bonnet.

▶ **Conjecture 1.1.** *Let $\mathcal{C}$ be a class of graphs of twin-width at most 3 such that there exists t such that no $G \in \mathcal{C}$ contains $K_{t,t}$ as a subgraph. Then $\mathcal{C}$ has bounded tree-width.*

For graph classes of twin-width at most one, we can easily argue that Conjecture 1.1 is true as follows. It is known [11] that if a graph class $\mathcal{C}$ is such that every $G \in \mathcal{C}$ has a contraction sequence in which red components (see Section 2) have bounded size, then $\mathcal{C}$ has bounded clique-width. Since in a contraction sequence of width one each red component has

size one, this implies that graph classes of twin-width one have bounded clique-width. In combination with the fact that graph classes of bounded clique-width which exclude $K_{t,t}$ as a subgraph have bounded tree-width, this proves the result.

However, for twin-width 2 or 3 we cannot assume that red components of trigraphs occurring in contraction sequences have bounded size, and so to attack Conjecture 1.1 one has to employ a more fine-tuned approach by directly analyzing contraction sequences.

Our contribution

We confirm Conjecture 1.1 for the case of graph classes of twin-width 2 and disprove the conjecture for graph classes of twin-width 3. Namely on the positive side we prove the following.

▶ **Theorem 1.2.** *Let t be an integer and let $\mathcal{C}$ be a class of graphs of twin-width at most 2 such that no $G \in \mathcal{C}$ contains $K_{t,t}$ as a subgraph. Then the tree-width of $\mathcal{C}$ is bounded by a polynomial function of t, namely at most $\mathcal{O}(t^{20})$.*

Theorem 1.2 allows us to apply the existing tools for bounded tree-width to sparse classes of twin-width 2, making many hard problems efficiently solvable on such classes. Moreover, Theorem 1.2 also leads to the following algorithmic corollary.

▶ **Corollary 1.3.** *Let $\mathcal{C}$ be a class of graphs such that there exists t such that no $G \in \mathcal{C}$ contains $K_{t,t}$ as a subgraph. Then there exists a constant c depending on $\mathcal{C}$, and a polynomial-time algorithm which for every $G \in \mathcal{C}$ either outputs a contraction sequence of G of width at most c, or correctly outputs that G has twin-width more than 2.*

On the negative side, we exhibit an example of a graph class $\mathcal{C}$ of twin-width 3 such that no $G \in \mathcal{C}$ contains a $K_{2,2} \simeq C_4$ subgraph, and that $\mathcal{C}$ has unbounded tree-width.

2 Related Definitions and Tools

We use standard graph-theoretic terminology and notation. All graphs considered in this paper are finite and simple, i.e., without loops and multiple edges. For the sake of completeness, we include the following folklore definition and tool.

▶ **Definition 2.1** (Tree-width [26]). *A* tree-decomposition *of a graph G is a pair (X, T) where T is a tree, whose vertices we call nodes, and $X = \{X_i \mid i \in V(T)\}$ is a collection of subsets of $V(G)$ such that*

1. $\bigcup_{i \in V(T)} X_i = V(G)$,
2. *for each edge $vw \in E(G)$, there is $i \in V(T)$ such that $v, w \in X_i$ and,*
3. *for each $v \in V(G)$ the set of nodes $\{i \mid v \in X_i\}$ forms a subtree of T.*

The width of tree-decomposition $(\{X_i \mid i \in V(T)\}, T)$ is equal to $max_{i \in V(T)}\{|X_i| - 1\}$. The tree-width *of a graph G is the minimum width over all tree-decompositions of G.*

For a graph G and $A, B \subseteq V(G)$, an $A - B$ path is a path with one endvertex in A and the other endvertex in B. We will use Menger's theorem (see for example [17]).

▶ **Theorem 2.2** (Menger's theorem). *Let G be a graph and $A, B \subseteq V(G)$. The minimum size of a set $S \subseteq V(G)$ such that there is no $A - B$ path in $G - S$ is equal to the maximum number of vertex-disjoint $A - B$ paths in G.*

For an integer N, the $N \times N$ *wall* (or *hexagonal grid*) is the graph consisting of N disjoint paths $P_1, \ldots, P_N$, each P_i with vertices $v_1^i, \ldots, v_N^i$ in this order in P_i, together with the edges given by the following rule: if both $i, j \in \{1, \ldots, N\}$ have the same parity and $i < N$, then v_j^i is adjacent to v_j^{i+1}.

For an integer N, a graph H is called an $N \times N$ *cubic mesh* if the maximum degree of H is 3 and the following holds; we can write $H = Q_1 \cup Q_2$, where each Q_i, $i = 1, 2$, is formed as a vertex-disjoint union of N paths, the ends of paths of Q_i are disjoint from $V(Q_{3-i})$ for $i = 1, 2$, and every component–path of Q_1 intersects every component–path of Q_2 in precisely one (common) subpath. These paths of Q_1 (resp. of Q_2) are called the *rows* (resp. *columns*) of the mesh H, and the vertices of H of degree 3 are called the *branching vertices* of H. The intersection of any row and any column of H must be a subpath of nonzero length since $\Delta(H) = 3$, and it contributes two branching vertices.

▶ **Observation 2.3.** *Any subdivision of the classical* $(2N + 2) \times (2N + 2)$ *wall (hexagonal grid) contains an* $N \times N$ *cubic mesh as a subgraph. See Figure 1.*

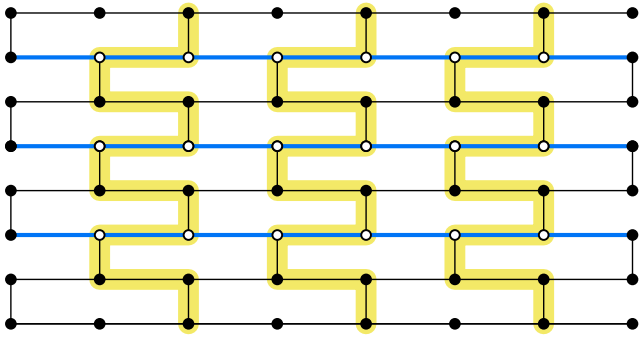


Figure 1 Example of an 8×8 wall. We obtain a 3×3 cubic mesh by considering the yellow paths as columns and the blue horizontal paths as rows. Hollow vertices are the branching vertices of this cubic mesh.

The notion of *twin-width* can in general be considered over arbitrary binary relational structures of a finite signature, but here we will define it and deal with it for only finite *simple graphs*. It is based on the following concept.

A *trigraph* is a simple graph G in which some edges are marked as *red*, and with respect to the red edges only, we naturally speak about *red neighbours* and *red degree* in G (while the terms neighbour and degree without "red" refer to all edges of G inclusive of red ones, and edges which are not red are also called *black*). 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 (said to *stem from* x_1, x_2) such that:

- the (full) neighbourhood of x_0 in G' (i.e., including the red neighbours), denoted by $N_{G'}(x_0)$, equals the union of the neighbourhoods $N_G(x_1)$ of x_1 and $N_G(x_2)$ of x_2 in G except x_1, x_2 themselves, that is, $N_{G'}(x_0) = (N_G(x_1) \cup N_G(x_2)) \setminus \{x_1, x_2\}$, and
- the red neighbours of x_0, denoted here by $N^r_{G'}(x_0)$, inherit all red neighbours of x_1 and of x_2 and add those in $N_G(x_1) \Delta N_G(x_2)$, that is, $N^r_{G'}(x_0) = \big(N^r_G(x_1) \cup N^r_G(x_2) \cup (N_G(x_1) \Delta N_G(x_2))\big) \setminus \{x_1, x_2\}$, where Δ denotes the symmetric set difference.

▶ **Definition 2.4** (Contraction sequence and twin-width [13]). *A* contraction sequence *of a trigraph G is a sequence of successive contractions turning G into a single vertex, and its* width *d is the maximum red degree of any vertex in any trigraph of the sequence. We say that G has or* admits a *d*-contraction sequence *if there is a contraction sequence of G of width at most d. The* twin-width *of a trigraph G is the minimum width over all possible contraction sequences of G.*

To define a contraction sequence and the twin-width of an ordinary graph G, we consider G as a trigraph with no red edges. In a summary, a graph has twin-width at most d, if and only if it admits a d-contraction sequence.

For our purpose, the following "inverted" view of a contraction sequence will be useful.

▶ **Definition 2.5** (Uncontraction sequence). *A* partitioned graph *is a graph G associated with an unordered vertex partition $\mathcal{P} = (P_1, \ldots, P_m)$ of $V(G)$. The* partitioned trigraph *of $(G, \mathcal{P})$ is a trigraph H on the vertex set $V(H) = \mathcal{P}$ such that $\{P_1, P_2\} \in E(H)$ if and only if G contains an edge from P_1 to P_2, and this edge $\{P_1, P_2\}$ is red if and only if* not *all pairs of $P_1 \times P_2$ are edges of G. An* uncontraction sequence *of an n-vertex graph G is a sequence of partitioned graphs $(G, \mathcal{P}^i)$ for $i = 1, \ldots, n$, where $\mathcal{P}^1 = \{V(G)\}$ and $\mathcal{P}^n$ is the partition of $V(G)$ into singletons, and for $1 < i \leq n$ the partition $\mathcal{P}^i$ is obtained from $\mathcal{P}^{i-1}$ by splitting an arbitrary one part of $\mathcal{P}^{i-1}$ into two.*

It is easy to observe that if $G_0, G_1, \ldots, G_{n-1}$ is a contraction sequence of the n-vertex graph $G = G_0$, then for the trigraph G_{n-k} (on k vertices) we may form the corresponding vertex partition $\mathcal{P}^k = (P_1^k, \ldots, P_k^k)$ of $V(G)$ by setting P_i^k to be the set of vertices of G contracted into $w_i \in V(G_{n-k})$, $i = 1, \ldots, k$. The partitioned trigraph of $(G, \mathcal{P}^k)$ is hence isomorphic to the trigraph G_{n-k} of the former contraction sequence, and these two possible approaches to twin-width in Definition 2.4 and Definition 2.5 exactly coincide.

3 Proof of Theorem 1.2

Our proof proceeds by contradiction. Thus, we assume that there is a $K_{t,t}$-free graph G of large tree-width which has twin-width at most 2. We then proceed in two steps:

- **Step I:** Since G has large tree-width, it has to contain a subdivision of a large wall, and hence a large cubic mesh, as a subgraph. Using this and the assumption that G has an uncontraction sequence of width at most 2, we show that there has to be a point in time during the uncontraction sequence such that there are four parts X_1, X_2, X_3, X_4 which form a red path in the corresponding trigraph and there are many disjoint paths of G fully contained in $X_1 \cup X_2 \cup X_3 \cup X_4$ with one endpoint in X_1 and the other in X_4.
- **Step II:** Using a carefully chosen invariant we show that the structure found in Step I can be maintained indefinitely during the subsequent steps of the uncontraction sequence (still of width at most 2). This yields a contradiction, since the final partition of the uncontraction sequence consists of singletons and there are no red edges.

So, we actually prove the following alternative formulation of Theorem 1.2.

▶ **Theorem 3.1.** *If G is a simple graph not containing a $K_{t,t}$ subgraph, but containing a subgraph (not necessarily induced) $H \subseteq G$ which is an $N \times N$ cubic mesh for $N = 16 \cdot (13t)^2$, then G is of twin-width at least 3.*

3.1 Proving Step I

We start with a trivial claim which is crucial in our arguments:

▶ **Observation 3.2.** *Assume a partitioned graph $(G, \mathcal{P})$ such that G has no $K_{t,t}$ subgraph, and $X_1, X_2 \in \mathcal{P}$ such that $|X_1|, |X_2| \geq t$. Then the partitioned trigraph of $(G, \mathcal{P})$ cannot contain a black edge $\{X_1, X_2\}$. In other words, whenever G has an edge from X_1 to X_2, there is a red edge $\{X_1, X_2\}$ in the partitioned trigraph of $(G, \mathcal{P})$.*

The first step in the proof of Theorem 1.2 is precisely formulated and proved next.

▶ **Lemma 3.3.** *Let t be an integer. Assume that G is a simple graph not containing a $K_{t,t}$ subgraph, but containing a subgraph (not necessarily induced) $H \subseteq G$ which is an $N \times N$ cubic mesh, where $N = 16 \cdot (13t)^2$. If there is an uncontraction sequence $(G, \mathcal{P}^i)$ for $i = 1, \ldots, |V(G)|$ of width at most 2, then there exists $m \in \{1, \ldots, |V(G)|\}$ such that the following holds:*

- *There are parts $X_1, X_2, X_3, X_4 \in \mathcal{P}^m$ with $|X_i| \geq t$ for each $i \in \{1, \ldots, 4\}$ which induce a red path in this order in the partitioned trigraph of $(G, \mathcal{P}^m)$, and there is no edge between X_1 and X_4.*
- *The set $X := X_1 \cup X_2 \cup X_3 \cup X_4$ induces in G a subgraph containing $s \geq 4t$ pairwise vertex-disjoint paths from X_1 to X_4.*

Proof. Set $k := 13t$. Let $m \in \{1, \ldots, |V(G)|\}$ be the least index such that every part of $\mathcal{P}^m$ contains less than $4k^2$ branching vertices of H. Then there is a part $Z \in \mathcal{P}^m$ such that Z contains at least $2k^2$ branching vertices of H (one of the two having resulted by the last splitting before $\mathcal{P}^m$). If Z intersected less than k rows and less than k columns of H, by the condition on intersecting rows and columns we would have less than $2k^2$ branching vertices in Z altogether. Hence, up to symmetry, we may assume that Z hits at least k rows of H. Moreover, since $N = 16 \cdot (13t)^2 = 16 \cdot k^2$, we have that Z contains branching vertices of less than $4k^2 = N/4$ rows of H.

Let $L_2, L_1, R_1, R_2 \in \mathcal{P}^m$ denote the (at most) four parts that are connected to Z by a red path of length ≤ 2 in the partitioned trigraph of $(G, \mathcal{P}^m)$. That is, we have a red path on, (L_2, L_1, Z, R_1, R_2) in this order (for simplicity, we silently ignore if some of the parts do not exist). Again, each of L_1, L_2, R_1, R_2 contains branching vertices of less than $4k^2 = N/4$ rows of H. Altogether, every row of H hit by Z has at least $2N - 5 \cdot N/4 = 3 \cdot N/4 > 0$ branching vertices not contained in $\bigcup \mathcal{Z}$ where $\mathcal{Z} = \{L_2, L_1, Z, R_1, R_2\}$. We have thus got, as subpaths of the rows hit by Z, a collection $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ of k vertex-disjoint paths in G which connect Z to parts in $\mathcal{P}^m \setminus \mathcal{Z}$. Let the paths in $\mathcal{Q}$ be chosen as inclusion-minimal and for any $Q_i \in \mathcal{Q}$ let a_i and b_i be the endpoints of Q_i.

We now proceed assuming that all L_1, L_2, R_1, R_2 have size at least t, the case when one of $\{L_1, L_2\}$ or one of $\{R_1, R_2\}$ is smaller than t is addressed below. Every path $Q \in \mathcal{Q}$, by its minimality, connects a vertex of Z to a vertex of $Y \in \mathcal{P}^m \setminus \mathcal{Z}$ where Y is a neighbour (red or black) of the set $\mathcal{Z}$ in the partitioned trigraph of $(G, \mathcal{P}^m)$. For any $X \in \{L_2, L_1, Z, R_1, R_2\}$, since $|X| \geq t$, the union of the parts adjacent to X by black edges in the partitioned trigraph is of cardinality less than t, or we have a $K_{t,t}$ subgraph. Together, at most $5t$ of the paths of $\mathcal{Q}$ in G may end in parts $Y \in \mathcal{P}^m \setminus \mathcal{Z}$ which are not red neighbours of $\mathcal{Z}$ in the partitioned trigraph. Since we have at most two red neighbours of parts of $\mathcal{Z}$ among the parts of $\mathcal{P}^m \setminus \mathcal{Z}$ (namely, the "outside" red neighbours of L_2 and of R_2), up to symmetry, the (unique) red neighbour $L_3 \in \mathcal{P}^m \setminus \mathcal{Z}$ of L_2 contains ends of at least $(k - 5t)/2 = 4t$ of the paths in $\mathcal{Q}$. In particular, L_3 has to exist, and L_3 is not adjacent to Z in the partitioned trigraph by Observation 3.2. We thus conclude by choosing $X_1 = L_3$, $X_2 = L_2$, $X_3 = L_1$ and $X_4 = Z$.

Next, we address the case when exactly one of $\{L_1, L_2\}$ or $\{R_1, R_2\}$ contains W such that $|W| < t$. Without loss of generality assume that $W \in \{R_1, R_2\}$ and so both L_1 and L_2 have size at least t. If $W = R_1$, set $\mathcal{Z}' := \{L_2, L_1, Z\}$, otherwise set $\mathcal{Z}' := \{L_2, L_1, Z, R_1\}$. Every path $Q \in \mathcal{Q}$, by its minimality, connects a vertex of Z to a vertex of $Y \in \mathcal{P}^m \setminus \mathcal{Z}'$ where Y is a neighbour (red or black) of the set $\mathcal{Z}'$ in the partitioned trigraph of $(G, \mathcal{P}^m)$. For any X in $\mathcal{Z}$, since $|X| \geq t$, the union of the parts adjacent to X by black edges in the partitioned trigraph is of cardinality less than t, or we have a $K_{t,t}$ subgraph. Together, at most $4t$ of the paths of $\mathcal{Q}$ in G may end in parts $Y \in \mathcal{P}^m \setminus \mathcal{Z}'$ which are not red neighbours of $\mathcal{Z}'$ in the partitioned trigraph, and at most t paths of Q can go through W. Ignoring all these at most $5t$ paths, all the remaining $|\mathcal{Q}| - 5t > 4t$ paths have to end in the red neighbor of L_2; call this neighbor L_3. We thus again conclude by choosing $X_1 = L_3$, $X_2 = L_2$, $X_3 = L_1$ and $X_4 = Z$.

Finally, we consider the case when there is $W_L \in \{L_1, L_2\}$ with $|W_L| < t$ and $W_r \in \{R_1, R_2\}$ with $|W_R| < t$. We will show that this leads to a contradiction. We first fix the choice of W_L and W_R more precisely. If both L_1 and L_2 have size less than t, then we choose L_1 as W_L and similarly, if both R_1 and R_2 have size less than t, we choose R_1 as W_R. Then in the path $L_2L_1ZR_1R_2$ all parts between W_L and W_R have size at least t. Since each part X between W_L and W_R has size at least t, the union of the parts adjacent to X by black edges has size less than t, as otherwise we have a $K_{t,t}$ as a subgraph. Thus, the union of W_L, W_R, and all black neighbors of (at most 3) parts between W_L an W_R has size at most $5t$. This means that there are at most $5t$ vertices which separate $\{a_1, \ldots, a_k\}$ from $\{b_1, \ldots, b_k\}$ (recall that for any $i \in [k]$ we denote by a_i and b_i the endpoints of Q_i). Since there are $k > 5t$ disjoint paths between $A := \{a_1, \ldots, a_k\}$ and $B := \{b_1, \ldots, b_k\}$, this is a contradiction to Menger's theorem. ◀

▶ **Observation 3.4.** *From Observation 3.2 it follows that each of the s paths from X_1 to X_4 claimed in Lemma 3.3 must also intersect X_2 and X_3 (possibly many times back and forth).*

3.2 Proving Step II

For our convenience, we introduce the following notation. Given a graph G, an uncontraction sequence $(G, \mathcal{P}^i)$ for $i = 1, \ldots, |V(G)|$, an integer $j \in \{1, \ldots, |V(G)|\}$, and $X \in \mathcal{P}^j$, we define $N_j^b(X)$ as the set of parts of $\mathcal{P}^j$ that have a black edge to X in the partitioned trigraph of $(G, \mathcal{P}^j)$; we call this set *the black neighborhood of X in $(G, \mathcal{P}^j)$*. Also, we set $\|N_j^b(X)\| := \sum_{Y \in N_j^b(X)} |Y|$ to be the total number of vertices of G in any part of the black neighborhood of X in $(G, \mathcal{P}^j)$.

▶ **Lemma 3.5.** *Let G be an arbitrary simple graph not containing a $K_{t,t}$ as a subgraph, and let $(G, \mathcal{P}^i)$ for $i = 1, \ldots, |V(G)|$ be an uncontraction sequence for G. Suppose that, for some $m \in \{1, \ldots, |V(G)|\}$, there are parts $X_1, X_2, X_3, X_4 \in \mathcal{P}^m$ with $|X_i| \geq t$ for each $i \in \{1, \ldots, 4\}$ such that:*

- *X_1, X_2, X_3, X_4 induce a red path in this order in the partitioned trigraph of $(G, \mathcal{P}^m)$,*
- *there is no edge between X_1 and X_4, and*
- *the set $X_1 \cup X_2 \cup X_3 \cup X_4$ induces in G a subgraph containing $s \geq 4t$ pairwise vertex-disjoint paths from X_1 to X_4.*

Then the width of this uncontraction sequence is greater than 2.

Proof. For a contradiction, suppose that the width of the considered uncontraction sequence of G is at most 2. We are going to formulate an invariant which is true for $(G, \mathcal{P}^m)$, and which will remain true at every subsequent step of the uncontraction sequence. Since this invariant, at the same time, will preclude the finest partition into singletons, the assumed sequence of width ≤ 2 cannot exist.

Invariant. At step $j \geq m$ of the uncontraction sequence, in the graph $(G, \mathcal{P}^j)$ and its partitioned trigraph, the following holds. There are 4 parts $X_1, X_2, X_3, X_4 \in \mathcal{P}^j$, each of size at least t, forming a red path in this order in the partitioned trigraph of $(G, \mathcal{P}^j)$, and parts X_1 and X_4 are not adjacent. Denote by s_j the maximum number of vertex-disjoint paths in $G[X_1 \cup X_2 \cup X_3 \cup X_4]$, starting in X_1 and ending in X_4. Then $s_j + \|N_j^b(X_2)\| + \|N_j^b(X_3)\| \geq 4t$.

Note that in the base case we have $s_m \geq 4t$, so the invariant is trivially satisfied for $j = m$. Also, without loss of generality we can assume that all the vertex-disjoint paths in $G[X_1 \cup X_2 \cup X_3 \cup X_4]$ are inclusion-wise minimal; so in particular, each path contains exactly one (starting) vertex in X_1 and exactly one (ending) vertex in X_4. See also an informal illustration in Figure 2.

Now suppose the invariant holds for some $j \in \{m, \ldots, |V(G)| - 1\}$ and let us prove that it is also preserved after the j-th uncontraction. First observe that for each $i \in \{1, 2, 3, 4\}$, G contains a bipartite clique with sides X_i and $\bigcup N_j^b(X_i)$. Since $|X_i| \geq t$ and G does not contain $K_{t,t}$ as a subgraph, we have $\|N_j^b(X_i)\| \leq t-1$. Therefore, $s_j \geq 4t-(t-1)-(t-1) \geq 2t$. Each of the s_j disjoint paths in $G[X_1 \cup X_2 \cup X_3 \cup X_4]$ must intersect each of the sets X_1, X_2, X_3, X_4 (Observation 3.4), so actually $|X_i| \geq s_j \geq 2t$ for each $i \in \{1, 2, 3, 4\}$.

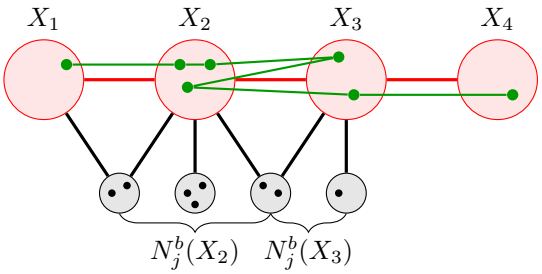


Figure 2 Illustration of 4 big parts X_1, X_2, X_3, X_4 forming red path (red), an example of one of s_j paths (green), and parts in black neighbourhoods $N_j^b(X_2)$, $N_j^b(X_3)$ (black).

We now analyze all the possible cases for an uncontraction step. Denote by $x \in \mathcal{P}^j$ the part that is split into new parts $y, z \in \mathcal{P}^{j+1}$. Also select some $s_j \geq 2t$ inclusionwise-minimal pairwise vertex-disjoint paths $P_1, \ldots, P_{s_j}$ from X_1 to X_4 in $G[X_1 \cup X_2 \cup X_3 \cup X_4]$. By minimality, Observation 3.2 and Observation 3.4, each path has its first vertex in X_1, its second vertex in X_2, its penultimate vertex in X_3, and its last vertex in X_4. Also notice that there cannot be any edges between X_1 and X_3, or any edges between X_2 and X_4 (otherwise, Observation 3.2 would apply and the red degree of X_2 or X_3 would be too large). Recall also there are no edges between X_1 and X_4.

If $x \notin \{X_1, X_2, X_3, X_4\}$, the black neighbourhood of X_2 and X_3 in the partitioned trigraph can only increase; also, $s_{j+1} = s_j$ as the vertex-disjoint paths in $G[X_1 \cup X_2 \cup X_3 \cup X_4]$ in $(G, \mathcal{P}^j)$ are preserved in the uncontraction step. Hence, the invariant is satisfied.

Suppose $x = X_1$ or $x = X_4$. Without loss of generality, $x = X_1$ (or suppose the red path is actually X_4, X_3, X_2, X_1), as illustrated in Figure 3a. Since at the j-th step, there were at least $s_j \geq 2t$ inclusion-wise minimal pairwise vertex-disjoint paths from X_1 to X_4 in $G[X_1 \cup X_2 \cup X_3 \cup X_4]$, at least t of these start in y or at least t of these start in z; again without loss of generality, assume that y contains at least t starts of the paths (so naturally $|y| \geq t$). As each of those paths have its second vertex in X_2, we know that yX_2 is an edge in the partitioned trigraph of $(G, \mathcal{P}^{j+1})$; and it must be a red edge, because both y and X_2 have size at least t and Observation 3.2 applies. We now claim that the red path y, X_2, X_3, X_4 preserves the invariant in $(G, \mathcal{P}^{j+1})$. Since $y, z \subseteq X_1$, both y and z are non-adjacent to X_3

and X_4. Observe also that z cannot be red-adjacent to X_2 in $(G, \mathcal{P}^{j+1})$ as in this case, the red degree of X_2 would be at least 3. However, z can be either black-adjacent or non-adjacent to X_2. If z is non-adjacent to X_2, then z cannot contain the start of any of the s_j paths P_i as the second vertex of each such path is in X_2. Hence we do not lose any path P_i by removing z from $X_1 \cup X_2 \cup X_3 \cup X_4$, so $s_{j+1} = s_j$ and the invariant still holds. If z is black-adjacent to X_2, then by a counting argument z contains at most $|z|$ starts of the considered paths P_i; hence, $s_{j+1} \geq s_j - |z|$. However, the number of vertices of G in the black neighborhood of X_2 also increases by $|z|$, because z was split out of X_1 which was a red neighbour of X_2. Hence the inequality from the invariant is satisfied:

$$s_{j+1} + \|N^b_{j+1}(X_2)\| + \|N^b_{j+1}(X_3)\| \geq (s_j - |z|) + (\|N^b_j(X_2)\| + |z|) + \|N^b_j(X_3)\| \geq 4t.$$

It is also easy to verify the remaining conditions of the invariant.

It remains to consider $x = X_2$ or $x = X_3$. Without loss of generality, $x = X_2$ (or suppose the red path $X_4, \ldots, X_1$). We first prove that at least one of y, z is red-adjacent to X_1 in $(G, \mathcal{P}^{j+1})$. Assume otherwise; then y is either non-adjacent to X_1 (and then $|N_G(X_1) \cap y| = 0$) or black-adjacent to X_1 (and then $|y| \leq t-1$ by Observation 3.2, so $|N_G(X_1) \cap y| \leq t-1$). Analogously, $|N_G(X_1) \cap z| \leq t-1$. Since $X_2 = y \cup z$, we conclude that $|N_G(X_1) \cap X_2| \leq 2t-2$. But each of $s_j \geq 2t$ vertex-disjoint paths $P_1, \ldots, P_{s_j}$ have two consecutive vertices in X_1 and X_2, so $|N_G(X_1) \cap X_2| \geq s_j$ – a contradiction. Hence at least one of y, z is red-adjacent to X_1. Repeating the same argument with X_3 instead of X_1 implies that at least one of y, z is red-adjacent to X_3. Additionally, X_3 is already red-adjacent to X_4, so exactly one of y, z is red-adjacent to X_3. Without loss of generality, assume that y is red-adjacent to X_3. We now consider two separate cases, depending on whether y is red-adjacent to X_1.

First assume that y is red-adjacent to X_1, as illustrated in Figure 3b. We claim that the red path X_1, y, X_3, X_4 preserves the invariant in $(G, \mathcal{P}^{j+1})$. Observe that z cannot be red-adjacent to y due to the red degree condition of y in $(G, \mathcal{P}^{j+1})$. If z is non-adjacent to y and X_3, no inclusion-wise minimal path from X_1 to X_4 in $G[X_1 \cup X_2 \cup X_3 \cup X_4]$ can be routed through z as all neighbors of z in $X_1 \cup X_2 \cup X_3 \cup X_4$ are in X_1. Then $s_{j+1} = s_j$ and it is easy to verify the remaining parts of the invariant. Next suppose z is black-adjacent to X_3; then by Observation 3.2 we have $|z| \leq t-1$, hence $|y| = |X_2| - |z| \geq t$. Also observe that at most $|z|$ paths P_i intersect z, so $s_{j+1} \geq s_j - |z|$. Therefore, the inequality from the invariant is preserved:

$$s_{j+1} + \|N^b_{j+1}(X_2)\| + \|N^b_{j+1}(X_3)\| \geq (s_j - |z|) + \|N^b_j(X_2)\| + (\|N^b_j(X_3) + |z|)\| \geq 4t,$$

and the remaining conditions of the invariant are easy to verify. Finally suppose z is non-adjacent to X_3 and black-adjacent to y. Then each path P_i must intersect y, hence $|y| \geq s_j > t$. Moreover, $s_{j+1} \geq s_j - |z|$ and again $\|N^b_{j+1}(X_2)\| \geq \|N^b_j(X_2)\| + |z|$, so the inequality from the invariant is preserved; once again, the remaining parts of the invariant follow.

It remains to consider the case where y is not red-adjacent to X_1 (i.e., either non-adjacent or black-adjacent to X_1), as illustrated in Figure 3c. We claim that z, y, X_3, X_4 is a red path preserving the invariant in $(G, \mathcal{P}^{j+1})$. We first prove that $|y|, |z| \geq t$. First assume that $|y| \leq t-1$. Then $|z| = |X_2| - |y| \geq 2t - (t-1) \geq t$, so by Observation 3.2, z is not black-adjacent to X_3; and it is not red-adjacent to X_3 as X_3 is already red-adjacent to y and X_4. Thus z is non-adjacent to X_3, so each path P_i must intersect y (this is because $X_2 = y \cup z$ and each path P_i contains an edge between X_2 and X_3). But then $|y| \geq s_t > t-1$ – a contradiction. Similarly, if $|z| \leq t-1$, then $|y| = |X_2| - |z| \geq t$, so by Observation 3.2, y is not black-adjacent to X_1; and it is not red-adjacent to X_1 by our assumption. So y is

non-adjacent to X_1 and each path P_i must intersect z – a contradiction. Hence, $|y|, |z| \geq t$. Applying Observation 3.2 three times, we find that y is non-adjacent to X_1; z is non-adjacent to X_3; and y is not black-adjacent to z. Since y must be adjacent to z (otherwise X_1 and X_4 would be in separate connected components of $G[X_1 \cup X_2 \cup X_3 \cup X_4]$), we get that y is red-adjacent to z. Hence the subgraph of the partitioned trigraph of $(G, \mathcal{P}^{j+1})$ induced by $\{X_1, z, y, X_3, X_4\}$ contains red edges X_1z, zy, yX_3 and X_3X_4 and no black edges. Therefore, the second vertex of each path P_i is actually in z; so we can remove a prefix from each path P_i so that each path starts in z, finishes in X_4 and is contained in $z \cup y \cup X_3 \cup X_4$. This witnesses that there exist s_j vertex-disjoint paths from z to X_4 in $G[z \cup y \cup X_3 \cup X_4]$, so $s_{j+1} \geq s_j$. Moreover, $\|N^b_{j+1}(y)\| = \|N^b_j(X_2)\|$ (as the black neighborhoods of y and z in $(G, \mathcal{P}^{j+1})$ are equal to the black neighborhood of X_2 in $(G, \mathcal{P}^j)$), and $\|N^b_{j+1}(X_3)\| = \|N^b_j(X_3)\|$ (as the black neighborhood of X_3 did not change during the uncontraction). We conclude that $s_{j+1} + \|N^b_{j+1}(y)\| + \|N^b_{j+1}(X_3)\| \geq 4t$, as required, and all the satisfaction of the remaining parts of the invariant is clear.

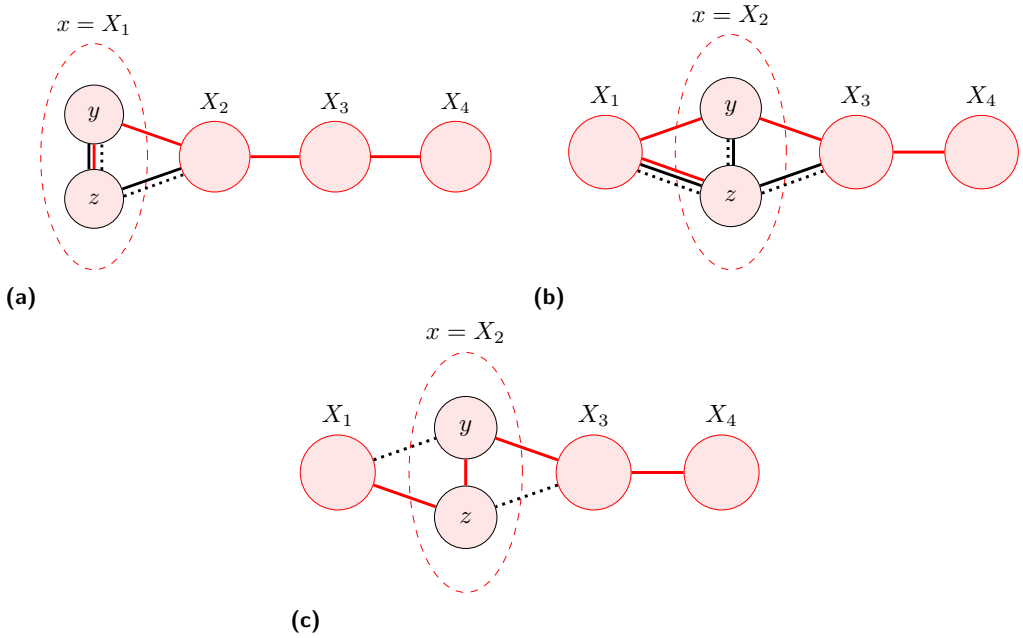


Figure 3 Illustration of possible uncontractions from the proof of Lemma 3.5. Multiple edge connections between parts show all possible edges, dotted edge means non-adjacency.

◀

3.3 Concluding the Main Proof

Observe that the basic assumptions of Theorem 3.1 are the same as those of Lemma 3.3, and the assumptions of Lemma 3.5 follow from Lemma 3.3. Hence, as detailed in the formal proof below, Lemma 3.3 and Lemma 3.5 together imply Theorem 3.1.

To extend this to a proof of Theorem 1.2, we use the following tool – an excluded-grid theorem for tree-width in the currently strongest published formulation (modulo polylog factors which we have "rounded up" for simplicity):

▶ **Theorem 3.6** (Chuzhoy and Tan [16]). *There is a function $f(n) \in \mathcal{O}(n^{10})$ such that, for every positive integer N, if a graph G is of tree-width at least $f(N)$, then G contains a subdivision of the $N \times N$ wall as a subgraph.*

Proof of Theorem 1.2. Let f be the function of Theorem 3.6, and choose $f_1(t) = f(2 \cdot 16 \cdot (13t)^2 + 2) \in \mathcal{O}(t^{20})$. If our graph G has tree-width at least $f_1(t)$, then G contains a subdivision of the $(2 \cdot 16 \cdot (13t)^2 + 2) \times (2 \cdot 16 \cdot (13t)^2 + 2)$ wall by Theorem 3.6. Consequently, by Observation 2.3, G contains a subgraph which is a $(16 \cdot (13t)^2) \times (16 \cdot (13t)^2)$ cubic mesh.

We now invoke Lemma 3.3, assuming G admits an uncontraction sequence of width at most 2; so, we obtain the parts X_1, X_2, X_3, X_4 as claimed by Lemma 3.3 and required by Lemma 3.5. By the subsequent invocation of Lemma 3.5, we immediately arrive at a contradiction to having the uncontraction sequence of width at most 2. Consequently, the tree-width less than $f_1(t) = f(2 \cdot 16 \cdot (13t)^2 + 2) \in \mathcal{O}(t^{20})$. ◀

Proof of Corollary 1.3. Let t be an integer and $\mathcal{C}$ a graph class such that no $G \in \mathcal{C}$ contains $K_{t,t}$ as a subgraph. Let f_1 be the polynomial function from (the proof of) Theorem 1.2 and set $k := f_1(t)$, which is a constant depending on $\mathcal{C}$. Let $G \in \mathcal{C}$ be the input graph. We first use the linear time algorithm of Bodlaender [6] to test whether G has tree-width at most k. If the answer is no, then we know by Theorem 1.2 that G cannot have twin-width at most 2. Otherwise, G has tree-width at most k, and we easily turn the decomposition into a branch-decomposition of width at most $k + 1$. This directly gives a boolean-width decomposition of width at most $k + 1$ [1] (with virtually the same decomposition tree). Finally, by the result of [14], from a boolean-width decomposition of G of width $k + 1$ one can obtain a contraction sequence of width at most $2^{k+2} - 1$ and an easy inspection of the proof shows that this can be done in polynomial time.

Thus, in the end (with respect to the fixed class $\mathcal{C}$) we compute in polynomial time a contraction sequence of G of width at most $2^{\text{poly}(t)}$. If the class $\mathcal{C}$ and implicit t were to be considered as parameters, the discussed algorithm would have an FPT runtime. ◀

3.4 Case of Twin-width 3

Lastly, we show that in the class of graphs of twin-width 3, no upper bound on the tree-width is possible even if we exclude $K_{2,2}$.

▶ **Lemma 3.7.** *For every positive integer N, there exists an $(N^2 + N)$-vertex graph with no $K_{2,2}$ subgraphs whose twin-width is at most 3 and tree-width is at least N.*

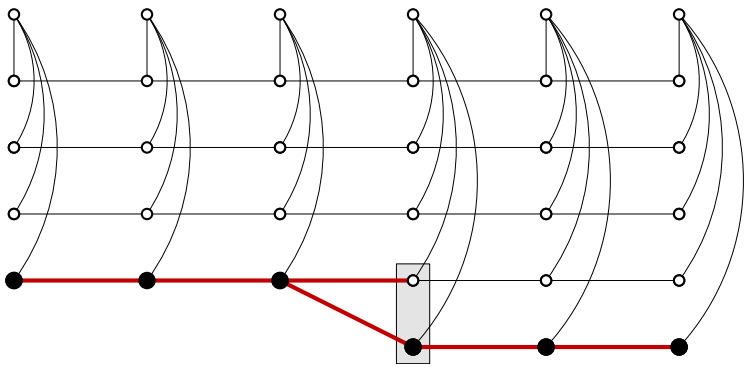


Figure 4 Illustration of the $K_{2,2}$-free graph of twin-width 3 and tree-width $\theta(\sqrt{n})$. It represents a partitioned trigraph of a contraction sequence of width 3: hollow vertices are singletons and the gray rectangle reveals the next contraction.

Proof. Let N be a positive integer. We take G the graph obtained from the disjoint union of N paths $P_1, \dots, P_N$ of length N, and for each $i \in [N]$, we add a new vertex adjacent to the i-th vertex of each path $P_1, \dots, P_N$. See Figure 4 for an illustration. Obviously G has $N^2 + N$ vertices and no $K_{2,2}$ as subgraph.

Observe that by contracting each path $P_1, \dots, P_N$, we obtain the complete bipartite graph $K_{N,N}$. As G admits $K_{N,N}$ as a minor, we deduce that its tree-width is at least N.

We claim that the twin-width of G is at most 3. We can assume without loss of generality that the edges of P_1 are red. We contract P_1 and P_2 by iteratively contracting their i-th vertices as illustrated in Figure 4. The maximum red degree of each encountered trigraph is 3 and we end up with a trigraph isomorphic to $G - P_2$. We can repeat this operation to end up with a trigraph isomorphic to $G - (P_2 \cup \dots \cup P_N)$. Then we can contract each pendant vertex with its unique neighbor to obtain a red path whose twin-width is obviously at most 2. We conclude that the twin-width of G is at most 3. ◀

4 Conclusions

We have shown that sparse classes of graphs of twin-width 2 have bounded tree-width. This means that the existing rich machinery of structural and algorithmic tools for tree-width is applicable to such graph classes.

One might wonder how one can relax the requirement on $\mathcal{C}$ being $K_{t,t}$-free to relate graphs of twin-width 2 to other well-studied graph notions. One could even try to drop any restrictions altogether and conjecture that any class of twin-width at most 2 has bounded clique-width. This cannot be true, since the class of unit interval graphs has twin-width 2 and unbounded clique-width [21]. However, we conjecture the following.

▶ **Conjecture 4.1.** *Let $\mathcal{C}$ be a stable class of graphs of twin-width at most 2. Then $\mathcal{C}$ has bounded clique-width.*

Here being stable means that there exists k such that no $G \in \mathcal{C}$ contains a half-graph of order k as a semi-induced subgraph. We refer to [20, 24] for details about stability and a discussion on how it relates to tree-width, clique-width and twin-width.

References

1. Isolde Adler, Binh-Minh Bui-Xuan, Yuri Rabinovich, Gabriel Renault, Jan Arne Telle, and Martin Vatshelle. On the boolean-width of a graph: Structure and applications. In *WG*, volume 6410 of *Lecture Notes in Computer Science*, pages 159–170, 2010.
2. Jungho Ahn, Kevin Hendrey, Donggyu Kim, and Sang-il Oum. Bounds for the twin-width of graphs. *CoRR*, abs/2110.03957, 2021. `arXiv:2110.03957`.
3. Jakub Balabán and Petr Hliněný. Twin-width is linear in the poset width. In *IPEC*, volume 214 of *LIPIcs*, pages 6:1–6:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
4. Jakub Balabán, Petr Hliněný, and Jan Jedelský. Twin-width and transductions of proper k-mixed-thin graphs. In *WG*, volume 13453 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2022.
5. Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is NP-complete. In *ICALP*, volume 229 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
6. Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. `doi:10.1137/S0097539793251219`.
7. Édouard Bonnet, Dibyayan Chakraborty, Eun Jung Kim, Noleen Köhler, Raul Lopes, and Stéphan Thomassé. Twin-width VIII: delineation and win-wins. In *IPEC*, volume 249 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

8 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *SODA*, pages 1977–1996. SIAM, 2021.

9 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In *ICALP*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

10 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Torunczyk. Twin-width IV: ordered graphs and matrices. In *STOC*, pages 924–937. ACM, 2022.

11 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. In *SODA*, pages 1036–1056. SIAM, 2022.

12 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. *Algorithmica*, 84(11):3300–3337, 2022. doi:10.1007/s00453-022-00965-5.

13 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *FOCS*, pages 601–612. IEEE, 2020.

14 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.

15 Édouard Bonnet, Jaroslav Nesetril, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. *CoRR*, abs/2102.06880, 2021. arXiv:2102.06880.

16 Julia Chuzhoy and Zihan Tan. Towards tight(er) bounds for the excluded grid theorem. *J. Comb. Theory, Ser. B*, 146:219–265, 2021. doi:10.1016/j.jctb.2020.09.010.

17 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

18 Jan Dreier, Jakub Gajarský, Yiting Jiang, Patrice Ossona de Mendez, and Jean-Florent Raymond. Twin-width and generalized coloring numbers. *Discret. Math.*, 345(3):112746, 2022. doi:10.1016/j.disc.2021.112746.

19 Jakub Gajarský, Michal Pilipczuk, Wojciech Przybyszewski, and Szymon Torunczyk. Twin-width and types. In *ICALP*, volume 229 of *LIPIcs*, pages 123:1–123:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

20 Jakub Gajarský, Michal Pilipczuk, and Szymon Torunczyk. Stable graphs of bounded twin-width. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2–5, 2022*, pages 39:1–39:12. ACM, 2022. doi:10.1145/3531130.3533356.

21 Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *Int. J. Found. Comput. Sci.*, 11(3):423–443, 2000. doi:10.1142/S0129054100000260.

22 Petr Hlinený, Filip Pokrývka, and Bodhayan Roy. FO model checking on geometric graphs. *Comput. Geom.*, 78:1–19, 2019. doi:10.1016/j.comgeo.2018.10.001.

23 Petr Hliněný and Jan Jedelský. Twin-width of planar graphs is at most 8, and at most 6 when bipartite planar. *CoRR*, abs/2210.08620, 2022. Accepted to ICALP 2023. arXiv:2210.08620.

24 Jaroslav Nesetril, Patrice Ossona de Mendez, Michal Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Rankwidth meets stability. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 2014–2033. SIAM, 2021. doi:10.1137/1.9781611976465.120.

25 Michal Pilipczuk, Marek Sokolowski, and Anna Zych-Pawlewicz. Compact representation for matrices of bounded twin-width. In *STACS*, volume 219 of *LIPIcs*, pages 52:1–52:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

26 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

Substring Complexity in Sublinear Space

Giulia Bernardini
University of Trieste, Italy

Gabriele Fici
Dipartimento di Matematica e Informatica, University of Palermo, Italy

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland

Solon P. Pissis
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands

Abstract

Shannon's entropy is a definitive lower bound for statistical compression. Unfortunately, no such clear measure exists for the compressibility of repetitive strings. Thus, ad hoc measures are employed to estimate the repetitiveness of strings, e.g., the size z of the Lempel–Ziv parse or the number r of equal-letter runs of the Burrows-Wheeler transform. A more recent one is the size γ of a smallest string attractor. Let T be a string of length n. A string attractor of T is a set of positions of T capturing the occurrences of all the substrings of T. Unfortunately, Kempa and Prezza [STOC 2018] showed that computing γ is NP-hard. Kociumaka et al. [LATIN 2020] considered a new measure of compressibility that is based on the function $S_T(k)$ counting the number of distinct substrings of length k of T, also known as the *substring complexity* of T. This new measure is defined as $\delta = \sup\{S_T(k)/k, k \geq 1\}$ and lower bounds all the relevant ad hoc measures previously considered. In particular, $\delta \leq \gamma$ always holds and δ can be computed in $\mathcal{O}(n)$ time using $\Theta(n)$ working space. Kociumaka et al. showed that one can construct an $\mathcal{O}(\delta \log \frac{n}{\delta})$-sized representation of T supporting efficient direct access and efficient pattern matching queries on T. Given that for highly compressible strings, δ is significantly smaller than n, it is natural to pose the following question:

Can we compute δ efficiently using sublinear working space?

It is straightforward to show that in the comparison model, any algorithm computing δ using $\mathcal{O}(b)$ space requires $\Omega(n^{2-o(1)}/b)$ time through a reduction from the element distinctness problem [Yao, SIAM J. Comput. 1994]. We thus wanted to investigate whether we can indeed match this lower bound. We address this algorithmic challenge by showing the following bounds to compute δ:

- $\mathcal{O}(\frac{n^3 \log b}{b^2})$ time using $\mathcal{O}(b)$ space, for any $b \in [1, n]$, in the comparison model.
- $\tilde{\mathcal{O}}(n^2/b)$[1] time using $\tilde{\mathcal{O}}(b)$ space, for any $b \in [\sqrt{n}, n]$, in the word RAM model. This gives an $\tilde{\mathcal{O}}(n^{1+\epsilon})$-time and $\tilde{\mathcal{O}}(n^{1-\epsilon})$-space algorithm to compute δ, for any $0 < \epsilon \leq 1/2$.

Let us remark that our algorithms compute $S_T(k)$, for all k, within the same complexities.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases sublinear-space algorithm, string algorithm, substring complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.12

Related Version *Full Version*: https://arxiv.org/abs/2007.08357

Funding *Giulia Bernardini*: MUR – FSE REACT EU – PON R&I 2014-2020.
Gabriele Fici: Projects MUR PRIN 2017 ADASCOML – 2017K7XPAN and MUR PRIN 2022 APML – 20229BCXNW.
Solon P. Pissis: Supported by the PANGAIA (No 872539) and ALPACA (No 956229) projects.

[1] The $\tilde{\mathcal{O}}(f)$ notation denotes $\mathcal{O}(f \cdot \text{polylog}(n))$.

1 Introduction

We are currently witnessing our world drowning in data. These datasets are generated by a large gamut of applications: databases, web applications, genome sequencing projects, scientific computations, sensors, e-mail, entertainment, and others. The biggest challenge is thus to develop theoretical and practical methods for processing datasets efficiently.

Compressed data representations that can be *directly* used in compressed form have a central role in this challenge [61]. Indeed, much of the currently fastest-growing data is highly repetitive; this, in turn, enables space reductions of orders of magnitude [35]. Prominent examples of such data include genome, versioned text, and software repositories collections. A common characteristic is that each element in a collection is very similar to every other.

Since a significant amount of this data is sequential, a considerable amount of algorithmic research has been devoted to text indexes over the past decades [68, 59, 29, 45, 31, 42, 44, 6, 23, 60, 46, 35, 47]. String processing applications (see [43, 2] for reviews) require fast access to the substrings of the input string. These applications rely on such text indexes, which arrange the string suffixes lexicographically in an ordered tree [68] or an ordered array [59].

This significant amount of research has resulted in compressed text indexes that support fast pattern searching in space close to the statistical entropy of the text collection. The problem, however, is that this kind of entropy is unable to capture repetitiveness [57, 58]. To achieve orders-of-magnitude space reductions, one thus needs to resort to other compression methods, such as Lempel-Ziv (LZ) [71], grammar compression [50] or run-length compressed Burrows-Wheeler transform (BWT) [35], to name a few; see [35] for a review.

Unlike Shannon's entropy, which is a definitive lower bound for statistical compression, no such clear measure exists for the compressibility of repetitive texts. Other than Kolmogorov's complexity [55], which is not computable, repetitiveness is measured in ad hoc terms, based on what the compressors may achieve. Such measures on a string T include: the number z of phrases produced by the LZ parsing of T; the size g of the smallest grammar generating T; and the number r of maximal equal-letter runs in the BWT of T. See [62] for a survey.

An improvement is the recent introduction of the *string attractor* [49] notion. Let T be a string of length n. An attractor Γ is a set of positions over $[1, n]$ such that any substring of T has an occurrence covering a position in Γ. The size γ of a smallest attractor asymptotically lower bounds all the repetitiveness measures listed above (and others; see [52]). Unfortunately, using indexes based on γ comes also with some challenges. Other than computing γ is NP-hard [49], it is unclear if γ is the definitive measure of repetitiveness: we do not know whether one can always represent T in $\mathcal{O}(\gamma)$ space (machine words). This motivated Christiansen et al. [18] to consider a new measure δ of compressibility, initially introduced in the area of string compression by Raskhodnikova et al. [66], and for which $\delta \leq \gamma$ *always holds* [18].

▶ **Definition 1** ([18]). *Let T be a string and $S_T(k)$ its* substring complexity*: the function counting the number of distinct substrings of length k of T. The* normalized substring complexity *of T is the function $S_T(k)/k$ and we set $\delta = \sup\{S_T(k)/k, k \geq 1\}$ its supremum.*

Christiansen et al. also showed that δ can be computed in $\mathcal{O}(n)$ time using $\Theta(n)$ working space. Kociumaka et al. [52, 53] showed that δ can also be strictly smaller than γ by up to a logarithmic factor: for any n and any δ, there are strings with $\gamma = \Omega(\delta \log \frac{n}{\delta})$. Moreover, Kociumaka et al. developed a representation of T of size $\mathcal{O}(\delta \log \frac{n}{\delta})$, which is worst-case optimal in terms of δ and allows for accessing any $T[i]$ in time $\mathcal{O}(\log \frac{n}{\delta})$ and for finding all occ occurrences of any pattern $P[1..m]$ in T in near-optimal time $\mathcal{O}(m \log n + occ \log^{\epsilon} n)$, for any constant $\epsilon > 0$ (see also [51] and [48] for further improvements). Since for highly compressible strings, δ is significantly smaller than n, we pose the following basic question:

Can we compute δ efficiently using sublinear working space?

The question on computing δ in *bounded space* arises naturally: it extends a large body of work on problems on strings, which admit a straightforward solution if we have the space to construct and store the suffix tree [68]; but as this is often not the case, one needs to overcome the space challenge by investigating space-time trade-offs for these problems.

Related Work. The standard approach for showing space-time trade-off lower bounds for problems answered in polynomial time has been to analyze their complexity on *(multi-way) branching programs*. In this model, the input is stored in read-only memory, the output in write-only memory, and neither is counted towards the space used by any algorithm. This model is powerful enough to simulate both Turing machines and standard RAM models that are unit-cost with respect to time and log-cost with respect to space. It was introduced by Borodin and Cook, who used it to prove that any multi-way branching program requires a time-space product of $\Omega(n^2/\log n)$ to sort n integers in the range $[1, n^2]$ [11, 4]. Unfortunately, the techniques in [11] yield only trivial bounds for problems with single outputs.

String algorithms that use sublinear space have been extensively studied over the past decades [36, 26, 64, 13, 67, 12, 54, 19, 34, 20, 22, 41, 40, 39, 38, 21, 37, 3, 63, 15, 65, 56]. The perhaps most relevant problem to our work is the classic longest common substring of two strings. Formally, given two strings X and Y of total length n, the *longest common substring* (LCS) problem consists in computing a longest string occurring as a substring of both X and Y. The LCS problem was conjectured by Knuth to require $\Omega(n \log n)$ time. This conjecture was disproved by Weiner who, in his seminal paper on suffix tree construction [68], showed how to solve the LCS problem in $\mathcal{O}(n)$ time for constant-sized alphabets. Farach showed that the same problem can be solved in the optimal $\mathcal{O}(n)$ time for polynomially-sized integer alphabets [29]. A straightforward space-time trade-off lower bound of $\Omega(b)$ space and $\Omega(n^2/b)$ time for the LCS problem can be derived from the problem of checking whether the length of an LCS is 0; i.e., deciding if X and Y have a common letter or not. Thus, in some sense, the LCS problem can be seen as a generalization of the *element distinctness* problem: given n elements over a domain D, decide whether all n elements are distinct.

On the upper bound side, Starikovskaya and Vildhøj showed that for any $b \in [n^{2/3}, n]$, the LCS problem can be solved in $\tilde{\mathcal{O}}(n^2/b)$ time and $\mathcal{O}(b)$ space [67]. In [54], Kociumaka et al. gave an $\mathcal{O}(n^2/b)$-time algorithm to find an LCS for any $b \in [1, n]$, and also provided a lower bound, which states that any deterministic multi-way branching program that uses $b \leq \frac{n}{\log n}$ space must take $\Omega(n\sqrt{\log(n/(b\log n))/\log\log(n/(b\log n))})$ time. This lower bound implies that the classic $\mathcal{O}(n)$-time solution for the LCS problem [68, 29] is optimal in the sense that we cannot hope for an $\mathcal{O}(n)$-time algorithm using $o(n/\log n)$ space. Unfortunately, we do not know if the $\mathcal{O}(b)$-space and $\mathcal{O}(n^2/b)$-time trade-off is generally the best possible for the LCS problem. For the easier element distinctness problem, Beame et al. [5] showed a randomized multiway branching program using $\tilde{\mathcal{O}}(n^{3/2}/\sqrt{b})$-time and $\mathcal{O}(b)$ space.

It is thus a big open question to answer whether the LCS problem can be solved asymptotically faster than $\mathcal{O}(n^2/b)$ using $\mathcal{O}(b)$ space. Towards this direction, Ben-Nun et al. exploited the intuition suggesting that an LCS of X and Y can be computed more efficiently when its length L is large [63] (see also [16]). The authors showed an algorithm which runs in $\tilde{\mathcal{O}}(\frac{n^2}{L\cdot b} + n)$ time, for any $b \in [1, n]$, using $\mathcal{O}(b)$ space. Still, a straightforward lower bound for the aforementioned problem is in $\Omega(\frac{n^2}{L^2\cdot b} + n)$ time when $\mathcal{O}(b)$ space is used; it seems that further insight is required to match this space-time trade-off lower bound.

Our Results and Techniques. Our goal is to efficiently compute δ using $\mathcal{O}(b)$ space. As a preliminary step towards this algorithmic challenge, we show the following theorem.

▶ **Theorem 2.** *Given a string T of length n, we can compute $\delta = \sup\{S_T(k)/k, k \geq 1\}$ in $\mathcal{O}(\frac{n^3 \log b}{b^2})$ time using $\mathcal{O}(b)$ space, for any $b \in [1, n]$, in the comparison model.*

It is straightforward to show that any comparison-based branching program to compute δ using $\mathcal{O}(b)$ space requires $\Omega(n^{2-o(1)}/b)$ time through a reduction from the element distinctness problem [70]. By Yao's lemma, this lower bound also applies to randomized branching programs [70]. This suggests that a natural intermediate step towards fully understanding the computation complexity of computing δ in small space should be designing an $\tilde{\mathcal{O}}(n^2/b)$-time algorithm using $\mathcal{O}(b)$ space (not necessarily in the comparison model).

The natural approach for computing δ is through computing all values of $S_T(k)$. In particular, this is the idea behind the straightforward $\mathcal{O}(n)$-time computation of δ using $\mathcal{O}(n)$ space [18]. It is unclear to us if a more direct approach exists (see also Section 6 for a combinatorial analysis on the behaviour of δ). Under this plausible assumption, we stress that computing $S_T(k)$, for all k one-by-one, is a more general problem than computing the length L of an LCS of X and Y, as an algorithm computing $S_T(k)$ can be used to compute L within the same complexities. This follows by the following argument: we compute $S_X(k)$, $S_Y(k)$, and $S_{X\#Y}(k)$ (where $\#$ is a special letter that does not occur in X or in Y) in parallel, and set L equal to the largest k such that $S_X(k) + S_Y(k) > S_{X\#Y}(k) - k$. As the best-known time upper bound for the very basic question of computing LCS in $\mathcal{O}(b)$ space remains to be $\mathcal{O}(n^2/b)$, this further motivates the algorithmic challenge of designing an algorithm with such bounds for computing δ. We address it by proving the following theorem.

▶ **Theorem 3.** *Given a string T of length n, we can compute $\delta = \sup\{S_T(k)/k, k \geq 1\}$ in $\tilde{\mathcal{O}}(n^2/b)$ time using $\tilde{\mathcal{O}}(b)$ space, for any $b \in [\sqrt{n}, n]$, in the word RAM model.*

Our algorithms compute $S_T(k)$, for all k, within the same complexities. To arrive at the $\mathcal{O}(\frac{n^3 \log b}{b^2})$-time bound, we split the computation of the values $S_T(k)$ in n/b phases: in each phase, we restrict to substrings whose length is in a range of size b. In turn, in each phase, we process the substrings that start within a range of b positions of T at a time, from left to right. With this scheme, we process in $\mathcal{O}(n \log b)$ time each block of n/b positions of T in each of the n/b phases, resulting in $\mathcal{O}(\frac{n^3 \log b}{b^2})$ time using $\mathcal{O}(b)$ space. For large enough b, we can process all the substrings of a single phase at once, saving a factor of n/b. We show in fact that a representation of all the occurrences of all the substrings of a phase can be packed in $\tilde{\mathcal{O}}(b)$ space if b is large enough, and process them in different ways depending on their period, following a scheme similar to [8]; we also adapt a method used in [9] to select a small set of anchors (length-b substrings), so that each fragment of T contains at least one anchor but their total number of occurrences in T is bounded. Note that Theorem 3 implies an $\tilde{\mathcal{O}}(n^{1+\epsilon})$-time and $\tilde{\mathcal{O}}(n^{1-\epsilon})$-space algorithm to compute δ, for any $0 < \epsilon \leq 1/2$.

Paper Organization. Section 2 introduces the basic definitions and notation we use and the space-time trade-off lower bound for computing δ. In Section 3, we present a simple $\mathcal{O}(n^3/b)$-time and $\mathcal{O}(b)$-space algorithm, for any $b \in [1, n]$. This algorithm is refined to run in $\mathcal{O}(\frac{n^3 \log b}{b^2})$ time using $\mathcal{O}(b)$ space, for any $b \in [1, n]$, in Section 4. Our main result, the $\tilde{\mathcal{O}}(n^2/b)$-time and $\tilde{\mathcal{O}}(b)$-space algorithm, for any $b \in [\sqrt{n}, n]$, is presented in Section 5. In Section 6, we consider the notion of substring complexity from the combinatorial point of view; and in Section 7, we conclude this paper with a final remark on approximating δ.

2 Preliminaries

An *alphabet* Σ is a finite nonempty set of elements called *letters*. We fix throughout a *string* $T = T[1] \cdots T[n]$ of *length* $|T| = n$ over an ordered alphabet Σ. By ε we denote the *empty string* of length 0. For two indices $1 \leq i \leq j \leq n$, the (i, j)*-fragment* of T is an *occurrence* of the underlying *substring* $T[i\,..\,j] = T[i] \cdots T[j]$. A *prefix* of T is a fragment of T of the form $T[1\,..\,j]$ and a *suffix* of T is a fragment of T of the form $T[i\,..\,n]$. A prefix (resp. suffix) of T is *proper* if it is not equal to T. We let $T^r = T[n]T[n-1] \cdots T[1]$ denote the *reversal* of T.

A positive integer p is *a period* of a string T if $T[i] = T[j]$ whenever $i = j \pmod p$; we call *the period* of T, denoted by $\mathsf{per}(T)$, the smallest such p. A string T is said to be *strongly periodic* if $\mathsf{per}(T) \leq |T|/4$ and *periodic* if $\mathsf{per}(T) \leq |T|/2$. We call the lexicographically smallest cyclic shift of $T[1\,..\,\mathsf{per}(T)]$ the *(Lyndon) root* of T. Notice that if T is periodic, then the root of T is always a fragment of T (that is, it has an occurrence in T).

For every string t and every natural number ℓ, we define the ℓth *power* of t, denoted by t^ℓ, by $t^0 = \varepsilon$ and $t^k = t^{k-1}t$, for integer $k = [1, \ell]$. A *run* with (Lyndon) root t in a string T is a periodic fragment $T[i\,..\,j] = t[q\,..\,|t|]t^\beta t[1\,..\,\gamma]$, with $q, \gamma \in [1, |t|]$ and β a positive integer, such that both $T[i-1\,..\,j]$ and $T[i\,..\,j+1]$, if defined, have their smallest period larger than $|t|$; we say that $q \in [1, |t|]$ is the *offset* of the run $t[q\,..\,|t|]t^\beta t[1\,..\,\gamma]$ and that two runs with the same root are *synchronized* if they have the same offset. We represent a run $t[q\,..\,|t|]t^\beta t[1\,..\,\gamma]$ by its starting and ending positions (i, j) in T, its root t, and its offset q.

The *element distinctness* problem asks to determine if all the elements of an array A of size n are pairwise distinct. Yao showed that, in the comparison-based branching program model, the time required to solve the element distinctness problem using $\mathcal{O}(b)$ space is in $\Omega(n^{2-o(1)}/b)$ [70]. We show the following lower bound for computing δ in the same model.

▶ **Theorem 4.** *The time required to compute δ for a string T of length n using $\mathcal{O}(b)$ space in the comparison model is in $\Omega(n^{2-o(1)}/b)$.*

Proof. We reduce the element distinctness problem to computing δ in $\mathcal{O}(n)$ time as follows. Let A be the input array for the element distinctness problem. Further let $\#_1, \#_2, \ldots, \#_n$ be pairwise distinct elements not occurring in A. We set $T = A \cdot \#_1\#_2 \ldots \#_n$, with $|T| = 2n$, $\#_i \neq A[j]$, for all $i, j \in [1, n]$. Observe that $S_T(k)/k < n$, for all $k \geq 2$, and thus $\delta = S_T(1) = n + |\{A\}|$. Then A has a repeating element if and only if $\delta < 2n$. ◀

3 $\mathcal{O}(n^3/b)$ Time Using $\mathcal{O}(b)$ Space in the Comparison Model

We start with a warm-up lemma to guide the reader smoothly to the $\mathcal{O}(n^3/b)$-time algorithm.

▶ **Lemma 5.** *Given a string T of length n, we can compute $\delta = \sup\{S_T(k)/k, k \geq 1\}$ in $\mathcal{O}(n^3)$ time using $\mathcal{O}(1)$ space in the comparison model.*

Proof. Let us consider each $S_T(k)$ separately, for all $k \in [1, n]$.

Set $S_T(k) = 0$. For all $i \in [1, n]$, we increase $S_T(k)$ if $T[i\,..\,i+k-1]$ is the first occurrence in $T[1\,..\,i+k-1]$. To perform this we check whether $T[j\,..\,j+k-1] = T[i\,..\,i+k-1]$, for all $j \in [1, i-1]$. We employ any linear-time constant-space pattern matching algorithm [36, 26, 13] to do this check in $\mathcal{O}(n)$ time using $\mathcal{O}(1)$ space for a single i. The statement follows. ◀

We next generalize Lemma 5 by employing the following straightforward observation.

▶ **Observation 6.** *Let S be a substring of T. If S occurs at least twice in T, then every substring of S occurs at least twice in T; if S occurs only once in T, then any substring of T containing S as a substring occurs only once in T.*

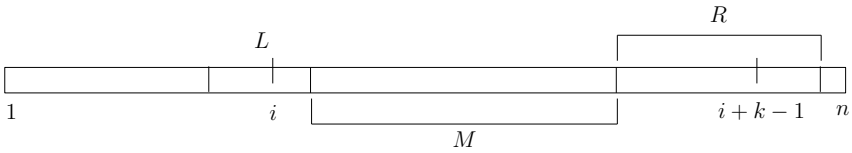


Figure 1 The main setting of the algorithm underlying Proposition 7.

Main Idea. Recall that we have $\mathcal{O}(b)$ budget for space. At any phase of the algorithm, we maintain $S_T(k)$ for b values of k, and iterate on consecutive non-overlapping substrings of T of length b, which we call *blocks*. This gives n/b phases and n/b iterations per phase, respectively. For each iteration, we define a substring M of T, which we call *anchor*. We search for occurrences of this anchor in T and extend each of the (at most) n occurrences of M in $\mathcal{O}(b)$ time per occurrence. This gives $\mathcal{O}(n^3/b)$ time and $\mathcal{O}(b)$ space.

▶ **Proposition 7.** *Given a string T of length n, we can compute $\delta = \sup\{S_T(k)/k, k \geq 1\}$ in $\mathcal{O}(n^3/b)$ time using $\mathcal{O}(b)$ space, for any $b \in [1, n]$, in the comparison model.*

Proof. Our algorithm consists of $n/b-2$ phases. In phase α, for all $\alpha \in [2, 3, \ldots, n/b-1]$,[2] we compute altogether the b values of $S_T(k)$, for all $k \in [\alpha b+1, (\alpha+1)b]$. Let $\mathcal{S} = \mathcal{S}[1\,..\,b]$ be an array of size b where we store the values of $S_T(k)$ corresponding to phase α: $\mathcal{S}[h] = S_T(\alpha b+h)$, for $h \in [1, b]$. At the end of phase α we maintain the maximum of $\mathcal{S}[h]/(\alpha b + h)$. Clearly, at the end of the whole procedure, we can output $\delta = \sup\{S_T(k)/k \mid k \geq 1\}$.

We start by decomposing T into n/b blocks $B_1, B_2, \ldots, B_{n/b}$, each of length b. We next describe our algorithm for a fixed phase $\alpha > 1$. First we set $\mathcal{S}[h] = 0$, for all $h \in [1, b]$. Let i be a position on T. For each k in the range of α, we want to know if $T[i\,..\,i+k-1]$ has its first occurrence in T at position i or if it occurs also at some position to the left of i. We process together all positions i in the same block $B_j = T[(j-1)b+1\,..\,jb]$, for every $j \in [1, n/b]$. Let $L = B_j$ be the block we are currently processing (inspect also Figure 1). To compute $S_T(k)$ we consider, for all $k \in [\alpha b + 1, (\alpha + 1)b]$, the length-$k$ fragments with starting position i in L. All such fragments share the same *anchor* $M = T[jb + 1\,..\,(j + \alpha - 1)b]$. The fragment of length k ends at position $i + k - 1$, which belongs to one of the *two blocks* succeeding M for all $k \in [\alpha b + 1, (\alpha + 1)b]$; we denote the concatenation of these two succeeding blocks as fragment R. In particular, we have $|M| = (\alpha - 1)b$ and $R = B_{j+\alpha}B_{j+\alpha+1}$.

We will use the occurrences of M in T that start before its *starting position* $jb + 1$ as anchors for finding possible occurrences of the length-k fragments starting within L. We search for such occurrences of M with any linear-time constant-space pattern matching algorithm [36, 26, 13]. For each such occurrence of M, we then need to check the b letters preceding it and the $2b - 1$ letters following it in order to determine whether it generates a previous occurrence of some $(i, i + k - 1)$-fragment, where i is a position within the block L. In particular, we check the b letters preceding it because L is the block of b positions preceding M; we check the $2b - 1$ letters following it because $k \leq (\alpha + 1)b$.

While processing $L = B_j = T[(j-1)b+1\,..\,jb]$, we also maintain an array $\mathsf{END}_L[1\,..\,b]$ of size b. After we have finished processing L, $\mathsf{END}_L[q]$ will store the length r_q of the longest prefix of R such that $T[(j-1)b+q\,..\,(j-1)b+q+|M|+r_q-1]$ occurs in T before

[2] We process the substrings of length $k \in [1, 2b]$ separately: for each block B_j, we compute an array LS_j of size b such that $\mathsf{LS}_j[q]$ is the length the *longest substring of length up to* $2b$ starting at position q in B_j that occurs in T before position $(j-1)b+q$. This is done as described in the proof of Lemma 8 and requires $\mathcal{O}(\frac{n^2 \log b}{b})$ total time. At the end of this procedure, we just maintain $\max\{S_T(k) \mid k \in [1, 2b]\}$.

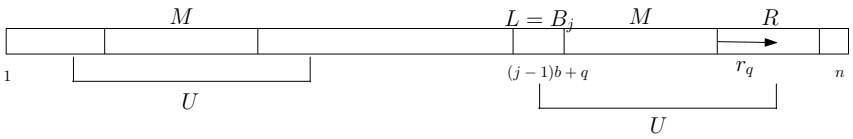


Figure 2 Largest r_q such that $U = T[(j-1)b+q\,..\,(j-1)b+q+|M|+r_q-1]$.

position $(j-1)b+q$ (inspect Figure 2). We compute END_L as follows. We search for all the occurrences of $M = T[jb+1\,..\,(j+\alpha-1)b]$ in $T[1\,..\,(j+\alpha-1)b-1]$, from left to right. Let $M = T[i'\,..\,i'+|M|-1]$ be one such occurrence. Let ℓ be the length of the longest common suffix of L and $T[1\,..\,i'-1]$; let r be the length of the longest common prefix of R and $T[i'+|M|\,..\,n]$. For each $q \geq b-\ell+1$, we update $\mathsf{END}_L[q]$ with the maximum between its previous value $\mathsf{END}_L[q]$ and r (note that we do not update any values if $\ell = 0$). After we have processed all the occurrences of M, for each q we increase by 1 all $S_T(k)$ such that $k > b-q+1+|M|+\mathsf{END}_L[q] = b-q+1+(\alpha-1)b+\mathsf{END}_L[q] = \alpha b-q+\mathsf{END}_L[q]+1$. This is an application of Observation 6: all these occurrences correspond to a substring that is longer than a substring that occurs *for the first time* in T at position $(j-1)b+q$.

The whole algorithm takes time $\mathcal{O}(\frac{n^3}{b})$: there are $\frac{n}{b}$ phases; in each phase, we consider $\frac{n}{b}$ blocks and for each block we spend $\mathcal{O}(n)$ time for pattern matching anchor M; for each occurrence of the anchor, we spend $\mathcal{O}(b)$ time for finding and updating the possible extensions, thus $\mathcal{O}(nb)$ time overall. We finally need $\mathcal{O}(b^2)$ time for updating the values of $S_T(k)$ for all k's in the range and all positions i in L. Overall this is $\mathcal{O}(\frac{n}{b}\cdot\frac{n}{b}\cdot(nb+b^2)) = \mathcal{O}(\frac{n^3}{b})$ time. ◀

4 $\mathcal{O}(\frac{n^3\log b}{b^2})$ Time Using $\mathcal{O}(b)$ Space in the Comparison Model

Recall that in Proposition 7, we spend $\mathcal{O}(nb+b^2)$ time to process the at most n occurrences of a single anchor M in T. We show here that all these occurrences can be processed in $\mathcal{O}(n\log b)$ time. This is made possible by processing together batches of occurrences of M that are *close enough* in T. This is done by means of answering longest common extension queries on suffix trees constructed for certain length-$\mathcal{O}(b)$ fragments of T.

The first trick is based on the following remark: The pattern matching algorithm for reporting the occurrences of M (e.g., [13]) reports the occurrences of M in real-time from left to right. Every such occurrence m of M is preceded by a block L' of length b on the left of m starting at position $m-b$ and ending at position $m-1$, and it is succeeded by a fragment R' of length $2b$ starting at position $m+|M|$ and ending at position $m+|M|+2b-1$. We thus need to find the longest common prefix of R and R' and the longest common suffix of L and L'. Let us describe the process for the longest common prefix of R and R'. (The procedure for the longest common suffix of L and L' is analogous and is executed simultaneously.)

We use the so-called *standard trick* to construct a sequence of $n/(4b)$ suffix trees for fragments of T of length $4b$ overlapping by $2b$ positions. We first concatenate each such fragment of length $4b$ with R. Constructing one such suffix tree takes $\mathcal{O}(b\log b)$ time using $\mathcal{O}(b)$ space [68]. Recall that an occurrence m of M implies an occurrence of R' at position $m+|M|$ and thus this position is part of some fragment of length $4b$. We preprocess this suffix tree in $\mathcal{O}(b)$ time and space to answer longest common prefix queries in $\mathcal{O}(1)$ time [7]. The whole preprocessing thus takes $n/(4b)\mathcal{O}(b\log b) = \mathcal{O}(n\log b)$ time. Thus, for any occurrence m of M we can find the longest right extension (and the longest left extension with a similar procedure) in $\mathcal{O}(1)$ time; recall that each extension cannot be of length greater than $2b$ so we do not miss any of them. To memorize the extensions we use an array END_L of size b.

For each occurrence of M, if we have a left extension of length $\ell > 0$ and a right extension of length r, we set $\mathsf{END}_L[b-\ell+1] = \max\{\mathsf{END}_L[b-\ell+1], r\}$ in $\mathcal{O}(1)$ time. At the end of this process we sweep through END_L and set $\mathsf{END}_L[q] = \max\{\mathsf{END}_L[q-1], \mathsf{END}_L[q]\}$, for all $i \in [2, b]$ by Observation 6: if we can extend a position q in L r positions to the right of M, then we must be able to extend position $q+1$ in L at least r positions to the right of M.

The second trick updates all values of $S_T(k)$ using array END_L in $\mathcal{O}(b)$ time instead of $\mathcal{O}(b^2)$ time. We use an array I of size b with all its entries initialized to 0; $I[h]$ will store the number of positions q in L such that the shortest unique substring starting at q is of length $\alpha b+h$. We fill in I scanning END_L: the shortest unique substring starting at q is by definition of length $\alpha b - q + \mathsf{END}_L[q] + 2$, which equals $\alpha b + h$ when $h = \mathsf{END}_L[q] - q + 2$. We thus increment $I[h]$ by one. We finally increase $S_T(\alpha b+h)$ by $\sum_{j=1}^{h} I[j]$ for all $h = 1, \ldots, b$. Thus, updating all values of $S_T(k)$ is implemented in $\mathcal{O}(b)$ time. We have arrived at Theorem 2.

5 $\tilde{\mathcal{O}}(\frac{n^2}{b})$ Time Using $\tilde{\mathcal{O}}(b)$ Space for $b \geq \sqrt{n}$ in the word RAM model

The algorithm underlying Theorem 2 is organized in n/b phases. In phase α we process b values of $S_T(k)$ making use of evenly-spaced fragments of T, each of length $(\alpha-1)b$, as *anchors* for finding possible multiple occurrences of the length-k fragments of T. Considering $\mathcal{O}(n/b)$ anchors in each phase and processing them one by one is the bottleneck of this algorithm. Our approach here is thus to avoid the burden of considering new anchors at every phase by carefully selecting a set of anchors that will remain *unchanged* in each phase of the algorithm. Let $c > 1$ be any integer constant. We will process the values of $S_T(k)$ for $k \leq cb$ (Section 5.1) and for $k > cb$ (Sections 5.2 to 5.5) in two different ways.

We work in the word RAM model and our goal is a deterministic algorithm. Recall that a suffix tree of any string of length d can be constructed in $\tilde{\mathcal{O}}(d)$ time using $\mathcal{O}(d)$ space [68, 29].

5.1 Computing $S_T(k)$ for Small k

We process together all values $k \in [1, cb]$. Like in Sections 3-4, for such values of k we split T into n/b blocks of b positions and work with each such block separately; we compute all values $S_T(k)$ and keep track of $\max_{k \leq cb} S_T(k)/k$ before computing $S_T(k)$ for all $k > cb$.

Consider block $L = B_j = T[(j-1)b+1\,..\,jb]$. We compute an array LS_L of size b such that $\mathsf{LS}_L[q]$ is the length the *longest substring* starting at position q in L that occurs in T before position $(j-1)b+q$, if this length does not exceed cb, otherwise we set it to ∞. This is done by constructing multiple generalized suffix trees of windows of length $2cb$ and L.

▶ **Lemma 8.** $\max_{k \leq cb} \frac{S_T(k)}{k}$ *can be computed in* $\tilde{\mathcal{O}}(n^2/b)$ *time and* $\mathcal{O}(b)$ *space, for any* $b \in [1, n]$.

Proof. We consider a block $L = T[(j-1)b+1\,..\,jb]$ of b positions of T at a time; for each position i of T within L, we must compute the length of the longest fragment $T[i\,..\,\ell]$ that occurs to the left of position i, if this length does not exceed cb. We consider windows of length $2cb$ over the prefix $T[1\,..\,(j+c)b]$, overlapping by cb positions. Clearly, if a fragment $T[i\,..\,\ell]$ occurs earlier in T, then it must be a substring of at least one such window. For a fixed L we initialize all the b positions of an array LS_L to 0; we then consider one window W of $2cb$ positions at a time, from left to right. At the end of the computation for a window W, $\mathsf{LS}_L[q]$ will store the length of the longest fragment starting at position $(j-1)b+q$ which occurs earlier in T. We proceed as follows to achieve this computation.

For the current window W of length $2cb$, we concatenate W and $T[(j-1)b+1\,..\,(j+c)b] = L \cdot T[jb+1\,..\,(j+c)b]$ (that is, block L and the following cb positions) constructing a new string S; we use a separator letter that does not occur in either of the two strings. We then

construct the suffix tree of S; and from there on the Longest Previous Factor (LPF) array of S in $\mathcal{O}(|S|) = \mathcal{O}(b)$ time [25]. The LPF array is an array of length $|S|$; for each position i of S, it gives the length of the longest substring of S that occurs both at i and to the left of i in S. Finally, we use this information to update the values of LS_L: $\mathsf{LS}_L[q]$ maintains the maximum between its previous value and the new value computed for the current W. We proceed to the next window. Once we have processed all the windows, we use LS_L to update the corresponding values of S_T in $\mathcal{O}(b)$ time the same way as we used END_L in Section 4.

The time and space complexity is as follows. There are n/b blocks in T, each of length b. For each such block, we consider $\mathcal{O}(n/b)$ windows of $2cb$ positions each, and for each window, we construct the suffix tree and the LPF array of the two underlying fragments of length $\mathcal{O}(b)$ in $\tilde{\mathcal{O}}(b)$ time using $\mathcal{O}(b)$ words of space. The whole procedure, for all n/b blocks and all $\mathcal{O}(n/b)$ windows, thus requires $\tilde{\mathcal{O}}(\frac{n}{b}\frac{n}{b}b) = \tilde{\mathcal{O}}(n^2/b)$ time using $\mathcal{O}(b)$ words of space. ◀

5.2 b-Runs and b-Gaps

When $k > cb$, we process b values of $S_T(k)$ at each phase, just like we did in Section 4. Different from Section 4, though, we aim at selecting a *global* set of anchors, carefully chosen among the length-b substrings of T. At each phase, we will distinguish three types of substrings, depending on the period of their length-b substrings. A *b-run* is a maximal fragment of length at least b such that each of its length-b substrings is strongly periodic; a standard reasoning based on the periodicity lemma [33] shows that the period of each b-run is at most $b/4$, and so a b-run is indeed a run. A *b-gap* is a maximal fragment such that none of its length-b substrings is strongly periodic. Any fragment of T of length at least b and period at most $b/4$ is fully contained in a unique b-run; and every fragment of T of length at least b and such that none of its length-b substrings is strongly periodic is fully contained in a unique b-gap. At each phase, the substrings to be processed are thus of three types: (i) either they are fully contained in a b-gap, or (ii) they are fully contained in a b-run, or (iii) neither of the two. We will process the substrings differently depending on their type. A standard reasoning using the periodicity lemma [33] shows that two b-runs cannot overlap by more than $b/2$ letters, so there are only $\mathcal{O}(n/b)$ of them. Lemma 10 states that we can identify and store the b-runs of T in such space complexity. For proving it we rely on the space-efficient construction of sparse suffix trees. The term "sparse" refers to constructing the compacted trie of an arbitrary subset of the set of the suffixes of the input string.

▶ **Theorem 9** ([10]). *Given a set $B \subseteq [n]$ of size $\Omega(\log n) \leq |B| \leq n$, there exists a deterministic algorithm which constructs the (sparse) suffix tree of B in $\mathcal{O}(n \log \frac{n}{|B|})$ time using $\mathcal{O}(|B|)$ words of space.*

▶ **Lemma 10.** *A representation of the b-runs of T can be computed in $\tilde{\mathcal{O}}(n)$ time using $\mathcal{O}(n/b)$ space, which is $\mathcal{O}(b)$ space when $b \geq \sqrt{n}$.*

Proof. We process windows of b positions of T at a time, with any two consecutive windows overlapping by $b/2$ positions. At each step, we compute the longest suffix, which has period at most $b/4$, of the window in $\mathcal{O}(b)$ time [24]. If such a suffix has nonzero length, we keep track of its starting position in T and extend it naïvely to the right as much as possible. If this extension results in a run of length at least b, we store its starting and ending position in a list ordered by starting position and resume the process using the window starting $b-1$ positions before the end of the run. Otherwise, if the extension results in a run shorter than b, we ignore it. Whenever we identify a b-run, we compute its root t in $\mathcal{O}(b)$ time [28], and store in a list the starting and ending position (s_r, e_r) of its root and the starting and

ending position (s, e) of the b-run (as mentioned above). After computing all b-runs in T, we construct the sparse suffix tree over the set of all s_r positions in the list. Each internal node of the sparse suffix tree, corresponding to a root of a b-run of T, is associated with the list of the starting and ending positions (s, e) of the b-runs corresponding to this root.

This procedure identifies all the b-runs of T. Indeed, consider a window $T[i\,..\,i+b-1]$. If a b-run Y with period $p \leq b/4$ begins between position i and position $i+b-1-p$, a prefix of it of length greater than p is a suffix of the window with period p. If it is the longest such suffix, it will be extended to the right allowing the identification of the whole Y. Otherwise, suppose there is a longer suffix of $T[i\,..\,i+b-1]$ with period $b/4 \geq p' > p$ (it cannot be $p' < p$, because otherwise, p' would have been the period of the whole suffix) that includes the whole prefix of Y in $T[i\,..\,i+b-1]$. In this case, we only extend the longer suffix and do not find Y at this stage. However, the longer suffix with period $p' \leq b/4$ is part of a run that overlaps with Y, and therefore such overlap must be shorter than $b/2$ because of the periodicity lemma [33]. This means: (a) this situation can only happen when the prefix of Y in $T[i\,..\,i+b-1]$ is shorter than $b/2$, thus a longer prefix of Y will be a suffix of the next window $T[i+b/2\,..\,i+3b/2-1]$; and (b) the period p' must break before the end of $T[i+b/2\,..\,i+3b/2-1]$, thus the prefix of Y in $T[i+b/2\,..\,i+3b/2-1]$ must be the longest suffix with period at most $b/4$ and will therefore be extended, allowing to identify the whole Y. Finally, if Y begins between position $i+b-p$ and position $i+b-1$ of $T[i\,..\,i+b-1]$, its prefix included in the window does not have a period p, and will therefore not be extended. However, the next window is $T[i+b/2\,..\,i+3b/2-1]$: since the length of any b-run is at least b, a prefix of the b-run of length greater than $b/2$ is now a suffix of the window, and since $p \leq b/4$, it will be extended to the right allowing the identification of the whole b-run.

The time and space complexity is as follows. We consider $\mathcal{O}(n/b)$ windows of length b. At each step, we spend $\mathcal{O}(b)$ time to compute the longest suffix of the current window with period at most $b/4$. Whenever we identify a suffix of a run Y with period at most $b/4$, we extend it naïvely to the right in $\mathcal{O}(|Y|)$ time, and the next window we consider only covers the last $b-1$ positions of Y. Since consecutive b-runs can only overlap by less than $b/2$ positions because of the periodicity lemma [33], they are at most $\mathcal{O}(n/b)$ and their total length is $\mathcal{O}(n)$, so it takes $\mathcal{O}(n)$ time to perform all extensions. For each b-run, we spend $\mathcal{O}(b)$ time to compute its root. For the sparse suffix tree, we employ Theorem 9. Hence the overall time complexity is $\tilde{\mathcal{O}}(n)$. As for the space, we process blocks of $\mathcal{O}(b)$ positions in $\mathcal{O}(b)$ space. We also store a pair of positions for each b-run, therefore the space required to store them is $\mathcal{O}(n/b)$, which is $\mathcal{O}(b)$ when $b \geq \sqrt{n}$. ◀

The output of Lemma 10 is a list representing all the b-runs of T in the natural left-to-right order. The b-gaps can be deduced from this list as follows: if $T[i\,..\,j]$ and $T[i'\,..\,j']$ are two consecutive b-runs in the list, then $T[j-b+2\,..\,i'+b-2]$ is a b-gap (if $T[i\,..\,j]$ is the first run, then so is $T[1\,..\,i+b-2]$, and similarly for the last run).

A subset of the length-b substrings of T is a *valid set of anchors* if two properties hold: (i) at least one anchor occurs in each fragment of T of length cb; and (ii) the total number of occurrences of all anchors in T is in $\mathcal{O}(n/b \cdot \log n)$. Lemma 11 shown next will be useful to prove that there always exists a set of valid anchors included in the b-gaps of T.

▶ **Lemma 11.** *Let Z be a string with all length-d substrings not strongly periodic, and $c > 1$ be any integer constant. Then we can compute in $\tilde{\mathcal{O}}(|Z|^2/d)$ time and $\tilde{\mathcal{O}}(|Z|/d + d)$ space a subset A of the length-d substrings of Z such that: (i) at least one $h \in A$ occurs in each fragment of Z of length cd; and (ii) the total number of occurrences of all $h \in A$ in Z is $\mathcal{O}(|Z|/d \cdot \log |Z|)$.*

Sketch of Proof. The high-level idea of the proof is to first reduce the problem to the following: we have $\mathcal{O}(|Z|/d)$ strings Z_i, each of length $5d/4$ and with all length-d substrings not strongly periodic, and a set of $\mathcal{O}(|Z|)$ possible anchors consisting of all length-d substrings of the Z_is. We want to choose a subset A of the anchors such that (i) at least one $h \in A$ occurs in each Z_i, (ii) the total number of occurrences of all $h \in A$ in the Z_is is $\mathcal{O}(|Z|/d \cdot \log |Z|)$. This is a special case of the Node Selection problem, considered in [9] as a strengthening of the well-known Hitting Set problem.[3] Indeed, we can take U to be the set of strings Z_i, V to be the set of possible anchors, and add an edge (u, v) in $G(U, V, E)$ when the possible anchor corresponding to v occurs in the string Z_i corresponding to u. Because every possible anchor is not strongly periodic and every Z_i is of the same length $5d/4$, the degree of every node $u \in U$ is $5d/4$. Then, by Lemma 5.4 of [9] (the weights are irrelevant) we can choose a set $V' \subseteq V$ such that (i) $N[u] \cap V' \neq \emptyset$ for every $u \in U$, (ii) $\sum_{u \in U} |N[u] \cap V'| = \mathcal{O}(|U| \log |U|) = \mathcal{O}(|Z|/d \cdot \log |Z|)$, so V' corresponds to a set of anchors A' with the sought properties. Furthermore, V' can be found in linear time and space in the size of G, which is $\mathcal{O}(|Z|)$. This is however not enough for our purposes, as we cannot store the whole G. Analysing the algorithm used inside the proof of Lemma 5.4 of [9] we see that it considers the nodes $v \in V$ one-by-one while maintaining some information of size $\mathcal{O}(|U|) = \mathcal{O}(|Z|/d)$ and a precomputed table of a size that can be bounded by the maximum degree of any $u \in U$, which is $\mathcal{O}(d)$. Furthermore, the algorithm accesses G only by iterating a constant number of times over the neighbours of the current node $v \in V$. In the full version, we show how to implement this efficiently in our model to achieve the claimed bounds. ◀

5.3 Processing the b-Gaps

For ease of presentation, in this section, we will assume that all length-b substrings of T are not strongly periodic, but no major changes are required to apply the same reasoning on the set of all b-gaps. Assume we have already computed a set A of valid anchors over T. For each $h \in A$, we compute a list of its occurrences in T. The overall size of these lists is $\mathcal{O}(n/b \cdot \log n)$ because of property (ii), and the occurrences of each $h \in A$ can be generated in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ space (plus the space to store the list) with any linear-time constant-space pattern matching algorithm, so $\tilde{\mathcal{O}}(n^2/b)$ time overall. We divide the computation of $S_T(k)$ in n/b phases. Consider phase α, in which we consider substrings of length $k \in [\alpha b + 1, (\alpha + 1)b]$. Because of property (i), at least one anchor occurs in the first cb positions of each such substring. We conceptually associate such a substring with the leftmost anchor $h \in A$ occurring therein, and we say that a fragment of T is *anchored* at an occurrence i of some anchor h if the leftmost occurrence of any anchor in the fragment is i. We then process the substrings according to the anchor with which they are associated.

All substrings associated with an anchor $h \in A$ have a (possibly empty) prefix of length $\mathcal{O}(b)$ where no anchors occur, followed by h and then by a suffix where any anchor can occur. This implies that any occurrence of such substrings can only start in a range of $\mathcal{O}(b)$ positions preceding some occurrence of h in T. In particular, if h occurs at position i in T and the closest anchor to its left is at position $i' < i$, the *starting range* of substrings of T associated with h is $[i' + 1, i]$, or $[1, i]$ if i is the first occurrence of any anchors in T. All starting ranges for all anchors can be computed in $\mathcal{O}(n)$ time by scanning the list of occurrences of the anchors. To update the values of $S_T(k)$ with the substrings associated with h we need to

[3] Let us remark that this problem has already been considered in the conference version [8], with a slightly different definition but essentially the same proof. However, our goal is a deterministic algorithm and to this end we need [9], the extended version of [8].

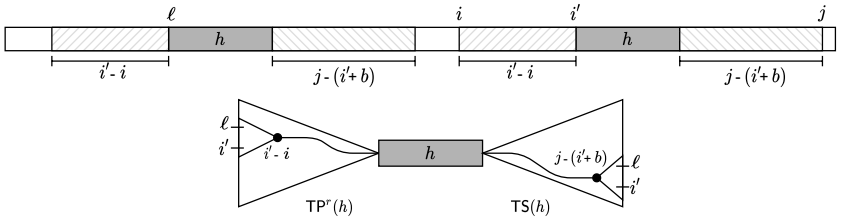


Figure 3 A previous occurrence of $T[i\,..\,j]$ anchored at i' can be detected using $D(h)$.

know, for each occurrence i of h in T and each of its previous occurrences $i' < i$, the longest left extension within the starting ranges of i and i', and the longest right extension of the fragments of T following the occurrences of h at i and i'. We cannot afford to store all these pairs of values explicitly as this would require $\tilde{\mathcal{O}}(n^2/b^2)$ space. We thus construct a separate data structure, denoted by $D(h)$, for each anchor $h \in A$. This data structure encode the same information in a compact form. We next describe the data structure and its construction.

$D(h)$ consists of two *compacted* tries $\mathsf{TP}^r(h)$ and $\mathsf{TS}(h)$. For every occurrence i of h in T, $\mathsf{TS}(h)$ contains a leaf corresponding to $T[i+b\,..\,n]$, and $\mathsf{TP}^r(h)$ a leaf corresponding to $(T[1\,..\,i-1])^r$, both labelled with position i. We only store the list of children and the length of the path label of each node, which we call its *depth*. Because of property (ii), the overall size of these data structures for all anchors is thus in $\mathcal{O}(n/b \cdot \log n)$. For any two occurrences i, i' of h, the depth of the lowest common ancestor of leaves i and i' in $\mathsf{TP}^r(h)$ gives the length of their longest left extension, and the depth of their lowest common ancestor in $\mathsf{TS}(h)$ gives the length of their longest right extension: see Figure 3 for an example. $D(h)$ can be efficiently constructed for all $h \in A$, as shown by Lemma 12.

▶ **Lemma 12.** *Data structures $D(h)$, for all $h \in A$, can be constructed in $\tilde{\mathcal{O}}(n^2/b)$ total time using $\tilde{\mathcal{O}}(n/b)$ space, which is $\tilde{\mathcal{O}}(b)$ when $b \geq \sqrt{n}$.*

Proof. Let $\mathsf{occ}(h)$ be the list of occurrences of anchor h in T, let $B = \bigcup_{h\in A} \mathsf{occ}(h)$ and $B' = \bigcup_{h\in A} \mathsf{occ}(h) + b - 1$. Recall that $D(h)$ consists of two *compacted* tries $\mathsf{TP}^r(h)$ and $\mathsf{TS}(h)$. We will first construct two global compacted tries $\mathsf{TP}^r(A)$ and $\mathsf{TS}(A)$ for all anchors in A, and then extract from them subtries $\mathsf{TP}^r(h)$ and $\mathsf{TS}(h)$ for each $h \in A$.

$\mathsf{TP}^r(A)$ and $\mathsf{TS}(A)$ are constructed in the same way, except that for $\mathsf{TP}^r(A)$ we consider the reversal of strings. To construct $\mathsf{TS}(A)$ we employ Theorem 9 on set B, as it is essentially the sparse suffix tree for the suffixes starting at positions in B; and to construct $\mathsf{TP}^r(A)$ we employ Theorem 9 on the reverse of T and set B'. Once we have constructed $\mathsf{TS}(A)$ and $\mathsf{TP}^r(A)$, to extract subtries $\mathsf{TS}(h)$ and $\mathsf{TP}^r(h)$ for $h \in A$ it suffices to spell h from the root of $\mathsf{TS}(A)$ (resp. h^r from the root of $\mathsf{TP}^r(A)$) and take the subtrie below.

The time and space complexity of computing $\mathsf{TS}(A)$ and $\mathsf{TP}^r(A)$ is as follows. The size of sets B and B' is $\mathcal{O}(n/b \cdot \log n) = \mathcal{O}(b \log n)$ when $b \geq \sqrt{n}$, thus by Theorem 9 we make use of $\mathcal{O}(b \log n)$ words of space. Again by Theorem 9, the overall time complexity to construct them is $\tilde{\mathcal{O}}(n)$. To find the right subtrie for each $h \in A$ we then spend $\tilde{\mathcal{O}}(b)$ time for each of the $\mathcal{O}(n/b \log n)$ anchors of A, thus again $\tilde{\mathcal{O}}(n)$ time overall. ◀

Computing $S_T(k)$ Using $D(h)$. Similar to Section 4, in each phase α we fill in an auxiliary array $I = I[1\,..\,b]$ such that, at the end of the phase, $I[q]$ contains the number of positions i in T such that the shortest substring that does not occur in T before position i is of length $\alpha b + q$. We proceed as follows. We consider one position of T at the time, from left to right. When we are at position i, let h be the leftmost anchor occurring at some position $i' \geq i$.

We binary search for the smallest position j such that $T[i\,..\,j]$ does not occur to the left of i using $D(h)$. We first identify in $\mathsf{TP}^r(h)$ the highest ancestor u of leaf i' with string depth at least $i'-i$. This corresponds to answering a *weighted level ancestor* query [30] on $\mathsf{TP}^r(h)$, where the weight of each node is its depth. After linear-time preprocessing, weighted ancestor queries for nodes of a weighted tree with integer weights from a universe $[1\,..\,U]$ can be answered in $\mathcal{O}(\log\log U)$ time [1]. In our case, the queries thus cost $\mathcal{O}(\log\log n)$ time.

We then start binary searching for the leftmost position j such that $T[i\,..\,j]$ does not occur to the left of position i and such that $|T[i\,..\,j]| \in [\alpha b+1,(\alpha+1)b]$: we thus look for j in the range $[i+\alpha b, i+(\alpha+1)b-1]$. For each value j considered in the binary search, we find in $\mathsf{TS}(h)$ the highest ancestor v of leaf i' with string depth at least $j-(i'+b)$, by answering a weighted level ancestor query. We then need to check whether $T[i\,..\,j]$ occurs somewhere to the left of i, in correspondence of a previous occurrence of anchor h, in which case we increase j in the next step; or it does not occur before, in which case we decrease j. We do so by looking at the leaves (occurrences of h) in the subtree below u in $\mathsf{TP}^r(h)$, denoted by $\mathsf{TP}^r(h)|u$, and in the subtree below v in $\mathsf{TS}(h)$, $\mathsf{TS}(h)|v$. Every leaf in the intersection of the two subsets of leaves corresponds to an occurrence of $T[i\,..\,j]$ in T. The information we need is whether i' is the smallest leaf in the intersection, meaning that $T[i\,..\,j]$ does not occur anywhere before. This reduces to a 2D range searching problem.

We assume that each leaf of each tree has a unique identifier, independent from their label and such that the identifiers of the leaves of any subtree form a contiguous range. For each leaf ℓ, its identifiers in $\mathsf{TP}^r(h)$ and $\mathsf{TS}(h)$ give the coordinates of a point on a plane, to which we assign ℓ as weight. By construction, the points corresponding to leaves in the intersection of $\mathsf{TP}^r(h)|u$ and $\mathsf{TS}(h)|v$ are contained in a rectangle: we need to find the point with the smallest weight there and check whether it is i' or not. Such queries can be answered in time $\mathcal{O}(\log s)$ with a data structure that is constructed in time and space $\mathcal{O}(s\log s)$, where s is the total number of points [17]. At the end of the binary search, if $j=i+\alpha b+q$ we increase the counter at $I[q]$ by one, unless $j=i+(\alpha+1)b-1$ and $T[i\,..\,j]$ occurs before i, in which case we do not increase any counters. We finally move to the next position of T.

▶ **Lemma 13.** *Assume that all length-b substrings of T are not strongly periodic. Then δ can be computed in $\tilde{\mathcal{O}}(n^2/b)$ time using $\tilde{\mathcal{O}}(n/b+b)$ space, which is $\tilde{\mathcal{O}}(b)$ when $b\geq\sqrt{n}$.*

Proof. Set A is selected in $\tilde{\mathcal{O}}(n^2/b)$ time and $\tilde{\mathcal{O}}(n/b+b)$ space as per Lemma 11, and $D(h)$ can be computed in the same time and space for all $h\in A$ and all phases, as per Lemma 12. In each phase α, we go over the n positions of T one at a time. At each position i we binary search for the shortest substring not occurring before i in $\mathcal{O}(\log\alpha b)$ steps, each requiring $\mathcal{O}(\log n)$ time. Over all $\mathcal{O}(n/b)$ phases, this requires $\tilde{\mathcal{O}}(n^2/b)$ time and $\tilde{\mathcal{O}}(n/b)$ space. ◀

5.4 Processing the b-Runs

Recall that we have computed, as per Lemma 10, a representation of all the b-runs of T. In this section, we only focus on the substrings of length at least b and periods at most $b/4$. Every occurrence of such a substring is fully contained in some b-run, and for ease of presentation we will assume that in phase α, in which we process substrings of length $k\in[\alpha b+1,(\alpha+1)b]$, every b-run is longer than αb. Observe that each substring of a b-run $T[s\,..\,e]$ with root t occurs also as a prefix of some fragment starting within the first $|t|$ positions of the run, which we call its *relevant* range. Since we aim to identify the leftmost occurrence of each substring of T, we can ignore all positions of a b-run after its relevant range. By slightly abusing notation, we select as anchors some fragments of the b-runs of T, instead of selecting substrings together with the whole set of their occurrences. However,

this set of anchors must have the following property, that for the anchors of Section 5.3 held naturally: for any two occurrences of the same substring in the relevant ranges, the leftmost occurrence of any anchor therein is at the same offset from the beginning of the substring. In phase α we use as anchors the first two occurrences of the root in each b-run: let H be this set of fragments of T. Clearly, H is of size $\mathcal{O}(n/b)$ because the representation of all the b-runs is of such size.

▶ **Lemma 14.** *For any two occurrences of the same substring of length at least b and period at most $b/4$, both starting in the relevant ranges of the b-runs of T, the leftmost occurrence of any $h \in H$ in each of them is at the same offset from the beginning of the substring.*

Proof. Let $Y = t[d\,..\,|t|]t^{\beta}t[1\,..\,f]$ be a fragment occurring at the first t positions in some b-run $T[s\,..\,e] = t[q\,..\,|t|]t^{\gamma}t[1\,..\,g]$. The anchors within $T[s\,..\,e]$ are, by definition, the occurrences of t at position $p_1 = s + (|t| - q + 1) \mod |t|$ and $p_2 = p_1 + |t|$. If $d \geq q$, the leftmost occurrence of any anchors in Y is at p_1, which is at offset $|t| - d + 2$ in Y. Otherwise, if $d < q$, the leftmost occurrence of any anchors in Y is at p_2, which is in any case at offset $|t| - d + 2$ in Y.

Consider another occurrence of Y in the first $|t|$ positions of some other b-run $T[s'\,..\,e'] = t[q'\,..\,|t|]t^{\gamma'}t[1\,..\,g']$. The anchors are the occurrences of t at position $p'_1 = s' + (|t| - q' + 1) \mod |t|$ and $p'_2 = p'_1 + |t|$; depending on whether $d \geq q'$ or not, the leftmost occurrence of any anchor in this occurrence of Y is either p'_1 or p'_2, in either case at offset $|t| - d + 2$ in Y. ◀

Let P be the set of roots of the b-runs of T. We construct a data structure $D(P)$ for all roots $t \in P$ similar to what we do in Section 5.3, but we use only the occurrences of t corresponding to fragments in H. We then proceed as described in Section 5.3 to fill in array I, except that, in each b-run with root t, we disregard any position after the first $|t|$.

We have arrived at the following lemma.

▶ **Lemma 15.** *The substrings of T that are fully contained within a b-run can be processed in $\tilde{\mathcal{O}}(n^2/b)$ time using $\mathcal{O}(n/b)$ space, which is $\mathcal{O}(b)$ when $b \geq \sqrt{n}$.*

5.5 Computing $S_T(k)$ for Large k

The occurrences of anchors $h \in A$ selected for the b-gaps anchor all the fragments fully contained in a b-gap and possibly some other fragments. However, we are not guaranteed that this holds for any fragment not fully contained in a b-run. Consider a fragment $T[i\,..\,j]$ of length at least b with period larger than $b/4$ (thus, not contained in any b-run) but containing a strongly periodic length-b fragment $T[i'\,..\,j']$ inside (so, not contained in any b-gap). Then, $T[i'\,..\,j']$ is fully contained in some b-run $T[s\,..\,e]$. Because $T[i\,..\,j]$ is not fully contained in $T[s\,..\,e]$, either $T[s-1\,..\,s+b-2]$ or $T[e-b+2\,..\,e+1]$ (that is, a length-b substring with exactly one letter before or after the b-run) is fully within $T[i\,..\,j]$. This suggest that we should augment A with the following length-b substrings: for each b-run $T[s\,..\,e]$, $T[s-1\,..\,s+b-2] \in A$ and $T[e-b+2\,..\,e+1] \in A$, and we consider all their occurrences in T. By the above reasoning, this guarantees that $T[i\,..\,j]$ contains an occurrence of some anchor inside. We are defining only $\mathcal{O}(n/b)$ new anchors, but then we need to consider all of their occurrences. Therefore, we need to argue that the total number of occurrences of the new anchors is $\mathcal{O}(n/b)$. It is enough to show this for the occurrences of the anchors $T[s-1\,..\,s+b-2]$, where the period of $T[s\,..\,s+b-2]$ is at most $b/4$. We claim that for any two such occurrences $T[s-1\,..\,s+b-2]$ and $T[s'-1\,..\,s'+b-2]$ with $s < s'$ we have $s + b/2 < s'$: otherwise $T[s\,..\,s+b-2]$ and $T[s'\,..\,s'+b-2]$ overlap by at least $b/2$ positions, but two b-runs cannot overlap by $b/2$ positions, a contradiction. We generate all

these occurrences and then process all the anchors as in Section 5.3. The only difference is the starting range associated with the anchors obtained from the suffix of some b-run: when they are not preceded by another anchor within cb positions, we take as starting range the αb positions preceding the anchor.

Let us put everything together. Before computing $S_T(k)$ in phases, we identify the b-runs and the b-gaps of T as per Lemma 10. We then extract a set of anchors from the b-gaps as described in Lemma 11, and we complement it with the length-b substrings that start one position before each b-run, and with the length-b substrings that end one position after the end of each b-run, to complete the set A of anchors. We then compute the list of occurrences of each $h \in A$; we also identify the relevant ranges within each b-run. We then proceed in phases. In each phase, we scan T from left to right and process all positions in b-gaps as per Section 5.3. All positions within a b-run are processed as per Section 5.4, and additionally as per Section 5.3, when they are within the starting range of an occurrence of some $h \in A$. At the end of a phase α, we have computed an auxiliary array I such that $I[h]$ gives the number of positions i of T such that the shortest substring that does not occur in T before position i is of length $\alpha b + h$. We use I to compute $S_T(k)$ for each $k \in [\alpha b + 1, (\alpha + 1)b]$ as in Section 4.

By combining Lemmas 13 and 15 we arrive at Theorem 3, the main result of this paper.

6 Substring Complexity from the Combinatorial Point of View

Knowing the substring complexity of a string can also be used to find other regularities. To mention a few, we have the following straightforward implications in sublinear working space:

- T has a substring of length k repeating in T if and only if $S_T(k) < n - k + 1$. This yields the length r of the longest repeated substring of T (also known as the repetition index of T) [68]. It is worth noticing that $S_T(k+1) = S_T(k) - 1$ for every $k > r$ [27] and that r approximates $\mathcal{O}(\log_{|\Sigma|} n)$ when T is randomly generated by a memoryless source [32].
- A string S is called a *minimal absent word* of T if S does not occur in T but all proper substrings of S occur in T. The length ℓ of a longest minimal absent word of T is equal to $2 + r$ [32]. This quantity is important because if two strings X and Y have the same set of distinct substrings up to length ℓ, then $X = Y$ [32, 14]. The length of a shortest absent word [69] of T over alphabet Σ is equal to the smallest k such that $S_T(k) < |\Sigma|^k$.
- The longest common substring of strings X and Y is equal to the largest k such that $S_X(k) + S_Y(k) > S_{X\#Y}(k) - k$, where $\#$ does not occur in X nor in Y, since there are precisely k distinct substrings of length k containing the letter $\#$ in $X\#Y$.

The substring complexity function is well studied in the area of combinatorics on words, both for finite and infinite strings. However, the normalization $S(k)/k$ and its supremum δ have not been considered until very recently. In [27] it is proved that the substring complexity $S_T(k)$ of a string T takes its maximum precisely for $k = R$, where R is the minimum length for which no substring of T has occurrences followed by different letters, and one has $S_T(R) = n + 1 - \max\{R, K\}$, where K is the length of the shortest unrepeated suffix of T. But this seems to be of little help in understanding the behaviour of the *normalized* substring complexity $S_T(k)/k$.

7 Approximating δ in Sublinear Space

Our algorithms compute the *exact value* of δ. If one is interested in a constant-factor approximation of δ (e.g., an algorithm's complexity has a polynomial dependency on δ [53]), then there is a simple algorithm in our model based on the following combinatorial observation, which follows directly by the number of fragments of length ℓ of a string of length n being $n - \ell + 1$, and by the fact that each fragment of length $\ell' > \ell$ has a prefix of length ℓ.

▶ **Observation 16.** *For any string T, let $S_T(k)$ be the number of distinct substrings of length k. The number $S_T(k')$ of distinct substrings of any length $k' > k$ is at least $S_T(k) - (k' - k)$.*

▶ **Lemma 17.** *Let $\delta' = \sup\{\frac{S_T(2^d)}{2^d} \mid d = 0, \ldots, \log n\}$. Then $\delta \leq 2\delta' + 1$.*

Proof. Let $\delta = \frac{S_T(k)}{k}$ for some $k \geq 1$, and let d be the integer such that

$$2^d \leq k < 2^{d+1}. \tag{1}$$

By the definition of δ', we have that $\delta' \geq \frac{S_T(2^{d+1})}{2^{d+1}}$. By applying Observation 16, we obtain:

$$\frac{S_T(2^{d+1})}{2^{d+1}} \geq \frac{S_T(k) - (2^{d+1} - k)}{2^{d+1}} \geq \frac{S_T(k) - (2^{d+1} - 2^d)}{2^{d+1}} \geq \frac{S_T(k)}{2k} - \frac{2^d}{2^{d+1}} = \frac{1}{2}\delta - \frac{1}{2}. \quad \blacktriangleleft$$

Recall that the algorithm underlying Theorem 2 works in $\frac{n}{b}$ phases, where each phase handles a range of b lengths k. By plugging in Lemma 17, the number of phases become $\Theta(\log n)$ – instead of $\Theta(n/b)$ – and so we obtain a simple $\tilde{\mathcal{O}}(n^2/b)$-time and $\mathcal{O}(b)$-space algorithm to approximate δ, within a constant factor, in the comparison model.

References

1. Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and static pattern matching. *ACM Trans. Algorithms*, 3(2):19, 2007. doi:10.1145/1240233.1240242.
2. Alberto Apostolico, Maxime Crochemore, Martin Farach-Colton, Zvi Galil, and S. Muthukrishnan. 40 years of suffix trees. *Commun. ACM*, 59(4):66–73, 2016. doi:10.1145/2810036.
3. Lorraine A. K. Ayad, Golnaz Badkobeh, Gabriele Fici, Alice Héliou, and Solon P. Pissis. Constructing antidictionaries in output-sensitive space. In *29th Data Compression Conference (DCC)*, pages 538–547, 2019. doi:10.1109/DCC.2019.00062.
4. Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20(2):270–277, 1991. doi:10.1137/0220017.
5. Paul Beame, Raphaël Clifford, and Widad Machmouchi. Element distinctness, frequency moments, and sliding windows. In *54th Symposium on Foundations of Computer Science (FOCS)*, pages 290–299, 2013. doi:10.1109/FOCS.2013.39.
6. Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In *46th Symposium on Theory of Computing, (STOC)*, pages 148–193, 2014. doi:10.1145/2591796.2591885.
7. Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *4th Latin American Symposium (LATIN)*, pages 88–94, 2000. doi:10.1007/10719839_9.
8. Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Even faster elastic-degenerate string matching via fast matrix multiplication. In *46th International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 132 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.21.
9. Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Elastic-degenerate string matching via fast matrix multiplication. *SIAM J. Comput.*, 51(3):549–576, 2022. doi:10.1137/20m1368033.
10. Or Birenzwige, Shay Golan, and Ely Porat. Locally consistent parsing for text indexing in small space. In *31st Symposium on Discrete Algorithms, (SODA)*, pages 607–626. SIAM, 2020. doi:10.1137/1.9781611975994.37.
11. Allan Borodin and Stephen A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.*, 11(2):287–297, 1982. doi:10.1137/0211022.
12. Dany Breslauer and Zvi Galil. Real-time streaming string-matching. *ACM Trans. Algorithms*, 10(4):22:1–22:12, 2014. doi:10.1145/2635814.
13. Dany Breslauer, Roberto Grossi, and Filippo Mignosi. Simple real-time constant-space string matching. *Theoret. Comput. Sci.*, 483:2–9, 2013. doi:10.1016/j.tcs.2012.11.040.

14 Arturo Carpi and Aldo de Luca. Words and special factors. *Theoret. Comput. Sci.*, 259(1-2):145–182, 2001. doi:10.1016/S0304-3975(99)00334-5.

15 Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern Hamming distances. In *52nd Symposium on Theory of Computing (STOC)*, pages 643–656, 2020. doi:10.1145/3357713.3384266.

16 Panagiotis Charalampopoulos, Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Linear-time algorithm for long LCF with k mismatches. In *29th Symposium on Combinatorial Pattern Matching (CPM)*, pages 23:1–23:16, 2018. doi:10.4230/LIPIcs.CPM.2018.23.

17 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988. doi:10.1137/0217026.

18 Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Trans. Algorithms*, 17(1):8:1–8:39, 2021. doi:10.1145/3426473.

19 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. Dictionary matching in a stream. In *23rd Annual European Symposium on Algorithms (ESA)*, pages 361–372, 2015. doi:10.1007/978-3-662-48350-3_31.

20 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k-mismatch problem revisited. In *37th Symposium on Discrete Algorithms (SODA)*, pages 2039–2052, 2016. doi:10.1137/1.9781611974331.ch142.

21 Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. In *30th Symposium on Discrete Algorithms (SODA)*, pages 1106–1125, 2019. doi:10.1137/1.9781611975482.68.

22 Raphaël Clifford and Tatiana Starikovskaya. Approximate Hamming distance in a stream. In *43rd International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 20:1–20:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.20.

23 Richard Cole, Tsvi Kopelowitz, and Moshe Lewenstein. Suffix trays and suffix trists: Structures for faster text indexing. *Algorithmica*, 72(2):450–466, 2015. doi:10.1007/s00453-013-9860-6.

24 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

25 Maxime Crochemore, Lucian Ilie, Costas S. Iliopoulos, Marcin Kubica, Wojciech Rytter, and Tomasz Walen. Computing the longest previous factor. *Eur. J. Comb.*, 34(1):15–26, 2013. doi:10.1016/j.ejc.2012.07.011.

26 Maxime Crochemore and Dominique Perrin. Two-way string matching. *J. ACM*, 38(3):651–675, 1991. doi:10.1145/116825.116845.

27 Aldo de Luca. On the combinatorics of finite words. *Theoret. Comput. Sci.*, 218(1):13–39, 1999. doi:10.1016/S0304-3975(98)00248-5.

28 Jean Pierre Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363–381, 1983.

29 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Symposium on Foundations of Computer Science (FOCS)*, pages 137–143, 1997. doi:10.1109/SFCS.1997.646102.

30 Martin Farach and S. Muthukrishnan. Perfect hashing for strings: Formalization and algorithms. In *7th Symposium on Combinatorial Pattern Matching (CPM)*, volume 1075 of *Lecture Notes in Computer Science*, pages 130–140. Springer, 1996. doi:10.1007/3-540-61258-0_11.

31 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

32 Gabriele Fici, Filippo Mignosi, Antonio Restivo, and Marinella Sciortino. Word assembly through minimal forbidden words. *Theoret. Comput. Sci.*, 359(1-3):214–230, 2006. doi:10.1016/j.tcs.2006.03.006.

33 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.2307/2034009.

34 Johannes Fischer, Travis Gagie, Pawel Gawrychowski, and Tomasz Kociumaka. Approximating LZ77 via small-space multiple-pattern matching. In *23rd European Symposium on Algorithms (ESA)*, pages 533–544, 2015. doi:10.1007/978-3-662-48350-3_45.

35 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):2:1–2:54, 2020. doi:10.1145/3375890.

36 Zvi Galil and Joel I. Seiferas. Time-space-optimal string matching. *J. Comput. Syst. Sci.*, 26(3):280–294, 1983. doi:10.1016/0022-0000(83)90002-8.

37 Pawel Gawrychowski and Tatiana Starikovskaya. Streaming dictionary matching with mismatches. In *30th Symposium on Combinatorial Pattern Matching (CPM)*, pages 21:1–21:15, 2019. doi:10.4230/LIPIcs.CPM.2019.21.

38 Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. The streaming k-mismatch problem: Tradeoffs between space and total time. In *31st Symposium on Combinatorial Pattern Matching (CPM)*, pages 15:1–15:15, 2020. doi:10.4230/LIPIcs.CPM.2020.15.

39 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Towards optimal approximate streaming pattern matching by matching multiple patterns in multiple streams. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 65:1–65:16, 2018. doi:10.4230/LIPIcs.ICALP.2018.65.

40 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Streaming pattern matching with d wildcards. *Algorithmica*, 81(5):1988–2015, 2019. doi:10.1007/s00453-018-0521-7.

41 Shay Golan and Ely Porat. Real-time streaming multi-pattern search for constant alphabet. In *25th Annual European Symposium on Algorithms (ESA)*, pages 41:1–41:15, 2017. doi:10.4230/LIPIcs.ESA.2017.41.

42 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005. doi:10.1137/S0097539702402354.

43 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

44 Wing-Kai Hon, Kunihiko Sadakane, and Wing-Kin Sung. Breaking a time-and-space barrier in constructing full-text indices. *SIAM J. Comput.*, 38(6):2162–2178, 2009. doi:10.1137/070685373.

45 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, 2006. doi:10.1145/1217856.1217858.

46 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *51st Symposium on Theory of Computing (STOC)*, pages 756–767, 2019. doi:10.1145/3313276.3316368.

47 Dominik Kempa and Tomasz Kociumaka. Breaking the O(n)-barrier in the construction of compressed suffix arrays and suffix trees. In *34th Symposium on Discrete Algorithms, SODA*, pages 5122–5202. SIAM, 2023. doi:10.1137/1.9781611977554.ch187.

48 Dominik Kempa and Tomasz Kociumaka. Collapsing the hierarchy of compressed data structures: Suffix arrays in optimal compressed space. *CoRR*, abs/2308.03635, 2023. doi:10.48550/arXiv.2308.03635.

49 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *50th Symposium on Theory of Computing (STOC)*, pages 827–840, 2018. doi:10.1145/3188745.3188814.

50 John C. Kieffer and En-Hui Yang. Grammar-based codes: a new class of universal lossless source codes. *IEEE Trans. Inf. Theory*, 46(3):737–754, 2000. doi:10.1109/18.841160.

51 Tomasz Kociumaka, Gonzalo Navarro, and Francisco Olivares. Near-optimal search time in δ-optimal space. In *15th Latin American Symposium (LATIN)*, volume 13568 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2022. doi:10.1007/978-3-031-20624-5_6.

52 Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Towards a definitive measure of repetitiveness. In *14th Latin American Symposium (LATIN)*, volume 12118 of *Lecture Notes in Computer Science*, pages 207–219. Springer, 2020. doi:10.1007/978-3-030-61792-9_17.

53 Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Toward a definitive compressibility measure for repetitive sequences. *IEEE Trans. Inf. Theory*, 69(4):2074–2092, 2023. doi:10.1109/TIT.2022.3224382.

54 Tomasz Kociumaka, Tatiana Starikovskaya, and Hjalte Wedel Vildhøj. Sublinear space algorithms for the longest common substring problem. In *22th European Symposium on Algorithms (ESA)*, pages 605–617, 2014. doi:10.1007/978-3-662-44777-2_50.

55 Andrei N. Kolmogorov. Three approaches to the quantitative definition of information. *International Journal of Computer Mathematics*, 2(1–4):157–168, 1968. doi:10.1080/00207166808803030.

56 Dmitry Kosolobov, Daniel Valenzuela, Gonzalo Navarro, and Simon J. Puglisi. Lempel-ziv-like parsing in small space. *Algorithmica*, 82(11):3195–3215, 2020. doi:10.1007/s00453-020-00722-6.

57 Sebastian Kreft and Gonzalo Navarro. On compressing and indexing repetitive sequences. *Theoret. Comput. Sci.*, 483:115–133, 2013. doi:10.1016/j.tcs.2012.02.006.

58 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of highly repetitive sequence collections. *J. Comput. Biol.*, 17(3):281–308, 2010. doi:10.1089/cmb.2009.0169.

59 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.

60 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In *28th Symposium on Discrete Algorithms (SODA)*, pages 408–424, 2017. doi:10.1137/1.9781611974782.26.

61 Gonzalo Navarro. *Compact Data Structures – A Practical Approach*. Cambridge University Press, 2016. URL: http://www.cambridge.org/de/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/compact-data-structures-practical-approach?format=HB.

62 Gonzalo Navarro. Indexing highly repetitive string collections, part I: repetitiveness measures. *ACM Comput. Surv.*, 54(2):29:1–29:31, 2022. doi:10.1145/3434399.

63 Stav Ben Nun, Shay Golan, Tomasz Kociumaka, and Matan Kraus. Time-space tradeoffs for finding a long common substring. In *31st Symposium on Combinatorial Pattern Matching (CPM)*, pages 5:1–5:14, 2020. doi:10.4230/LIPIcs.CPM.2020.5.

64 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *50th Symposium on Foundations of Computer Science (FOCS)*, pages 315–323, 2009. doi:10.1109/FOCS.2009.11.

65 Jakub Radoszewski and Tatiana Starikovskaya. Streaming k-mismatch with error correcting and applications. *Inf. Comput.*, 271:104513, 2020. doi:10.1016/j.ic.2019.104513.

66 Sofya Raskhodnikova, Dana Ron, Ronitt Rubinfeld, and Adam D. Smith. Sublinear algorithms for approximating string compressibility. *Algorithmica*, 65(3):685–709, 2013. doi:10.1007/s00453-012-9618-6.

67 Tatiana Starikovskaya and Hjalte Wedel Vildhøj. Time-space trade-offs for the longest common substring problem. In *24th Symposium on Combinatorial Pattern Matching (CPM)*, pages 223–234, 2013. doi:10.1007/978-3-642-38905-4_22.

68 Peter Weiner. Linear pattern matching algorithms. In *14th Symposium on Switching and Automata Theory*, pages 1–11, 1973. doi:10.1109/SWAT.1973.13.

69 Zong-Da Wu, Tao Jiang, and Wu-Jie Su. Efficient computation of shortest absent words in a genomic sequence. *Inf. Process. Lett.*, 110(14-15):596–601, 2010. doi:10.1016/j.ipl.2010.05.008.

70 Andrew Chi-Chih Yao. Near-optimal time-space tradeoff for element distinctness. *SIAM J. Comput.*, 23(5):966–975, 1994. doi:10.1137/S0097539788148959.

71 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.

New Support Size Bounds for Integer Programming, Applied to Makespan Minimization on Uniformly Related Machines

Sebastian Berndt
Institute for Theoretical Computer Science, University of Lübeck, Germany

Hauke Brinkop
Kiel University, Germany

Klaus Jansen
Kiel University, Germany

Matthias Mnich
Hamburg University of Technology, Institute for Algorithms and Complexity, Germany

Tobias Stamm
Hamburg University of Technology, Institute for Algorithms and Complexity, Germany

Abstract

Mixed-integer linear programming (MILP) is at the core of many advanced algorithms for solving fundamental problems in combinatorial optimization. The complexity of solving MILPs directly correlates with their support size, which is the minimum number of non-zero integer variables in an optimal solution. A hallmark result by Eisenbrand and Shmonin (*Oper. Res. Lett.*, 2006) shows that any feasible integer linear program (ILP) has a solution with support size $s \leq 2m \cdot \log(4m\Delta)$, where m is the number of constraints, and Δ is the largest absolute coefficient in any constraint.

Our main combinatorial result are improved support size bounds for ILPs.

We show that any ILP has a solution with support size $s \leq m \cdot (\log(3A_{\max}) + \sqrt{\log(A_{\max})})$, where $A_{\max} \coloneqq \|A\|_1$ denotes the 1-norm of the constraint matrix A. Furthermore, we show support bounds in the linearized form $s \leq 2m \cdot \log(1.46A_{\max})$. Our upper bounds also hold with $A_{\max}$ replaced by $\sqrt{m}\Delta$, which improves on the previously best constants in the linearized form.

Our main algorithmic result are the fastest known approximation schemes for fundamental scheduling problems, which use the improved support bounds as one ingredient.

We design an efficient approximation scheme (EPTAS) for makespan minimization on uniformly related machines ($Q||C_{\max}$). Our EPTAS yields a $(1+\varepsilon)$-approximation for $Q||C_{\max}$ on N jobs in time $2^{\mathcal{O}(1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon)))} + \mathcal{O}(N)$, which improves over the previously fastest algorithm by Jansen, Klein and Verschae (*Math. Oper. Res.*, 2020) with run time $2^{\mathcal{O}(1/\varepsilon \log^4(1/\varepsilon))} + N^{\mathcal{O}(1)}$. Arguably, our approximation scheme is also simpler than all previous EPTASes for $Q||C_{\max}$, as we reduce the problem to a novel MILP formulation which greatly benefits from the small support.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Integer programming, scheduling algorithms, uniformly related machines, makespan minimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.13

Related Version *arXiv Version*: `https://arxiv.org/abs/2305.08432` [7]

Funding *Hauke Brinkop*: Partially supported by DFG project `JA 612/25-1`, Fein-granulare Komplexität und Algorithmen für Scheduling und Packungen.
Klaus Jansen: Partially supported by DFG project `JA 612/25-1`, Fein-granulare Komplexität und Algorithmen für Scheduling und Packungen.
Matthias Mnich: Partially supported by DFG project `MN 59/4-1`, Multivariate algorithms for high-multiplicity scheduling.

1 Introduction

The Integer Linear Programming (ILP) problem is to find an optimal integral solution $\boldsymbol{x}^\star \in \mathbb{Z}^n_{\geq 0}$, which minimizes a linear objective $\boldsymbol{c}^\intercal \boldsymbol{x}$ subject to a system $A\boldsymbol{x} = \boldsymbol{b}$ of m linear constraints. In the Mixed-Integer Linear Programming (MILP) problem, the solution sought is a pair $(\boldsymbol{x}^\star, \boldsymbol{y}^\star) \in \mathbb{Z}^n_{\geq 0} \times \mathbb{Q}^r_{\geq 0}$, which minimizes $\boldsymbol{c}^\intercal(\boldsymbol{x}\ \boldsymbol{y})$ subject to $A\boldsymbol{x} + B\boldsymbol{y} = \boldsymbol{b}_1$ and $C\boldsymbol{y} = \boldsymbol{b}_2$. Solving (M)ILPs is at the core of many advanced algorithms for fundamental combinatorial optimization problems. Very often, the run time of these algorithms scales with the *support size* of the (M)ILPs, which is the smallest number of non-zero entries in an optimal solution $\boldsymbol{x}^\star$ resp. $(\boldsymbol{x}^\star, \boldsymbol{y}^\star)$. Thus, support size bounds have found applications all over computer science, for example in scheduling [5, 25], logic [30, 35], and even complexity theory [21]. Therefore, the smaller the support size, the better these results become. Hence, an important research direction is to prove strong upper bounds on the support size of (M)ILPs. The original result on support size bounds, which is already finding its way into the standard curriculum of integer programming courses [37, Lemma 6.1] is:

▶ **Proposition 1** (Eisenbrand, Shmonin [15, Thm. 1(ii)]). *Any feasible and bounded integer program with m constraints admits a solution with support size $s \leq 2m\log(4m\Delta)$, where Δ is the largest absolute value of any entry in the constraint matrix A.*

The result by Eisenbrand and Shmonin is about feasible solutions, and thus does not depend on any objective function. Aliev et al. [2] improved the Eisenbrand-Shmonin bound to $2m\log(2\sqrt{m}\Delta)$, and showed it to hold even for the support size of optimal solutions with respect to any linear objective function. Recently, Gribanov et al. [19] were the first to achieve a leading coefficient of 1, with a bound of the form $m\log(c_1 \cdot \sqrt{m}\Delta \cdot \sqrt{\log(c_2 \cdot \sqrt{m}\Delta)})$, and improved constants of $1.18m\log(7.02\sqrt{m}\Delta)$ in the linearized form. For the special cases of positively space spanning matrices [1], and in the average case over all right-hand sides [34], results on the order of $\mathcal{O}(m)$ have recently been obtained. One important application of the Eisenbrand-Shmonin bound are efficient polynomial-time approximation schemes (EPTAS) for scheduling problems [23]. An EPTAS computes, for any problem instance $\mathcal{I}$ and any $\varepsilon > 0$, a solution whose value is within $(1+\varepsilon)$ of the optimal solution value in time $f(1/\varepsilon)\langle\mathcal{I}\rangle^{\mathcal{O}(1)}$, where $\langle\mathcal{I}\rangle$ denotes the encoding size of $\mathcal{I}$. Several EPTAS were devised for the classical scheduling problem of makespan minimization on uniformly related machines. This problem is denoted as $Q||C_{\max}$ in Graham's 3-field notation [18]. The input to $Q||C_{\max}$ is a set $\mathcal{J}$ of N jobs, each of which is characterized by an integer processing time p_j, and a set $\mathcal{M}$ of M machines, each of which is characterized by an integer speed s_i. The goal is to find an assignment (or schedule) $\sigma : \mathcal{J} \to \mathcal{M}$ of jobs to machines, which minimizes the maximum completion time $C_{\max} := \max_{i\in\mathcal{M}} \sum_{j\in\sigma^{-1}(i)} p_j/s_i$ of any machine. Problem $Q||C_{\max}$ is well-known to be NP-hard, even in the special case of unit speeds $s_1 = \cdots = s_m$, but still approximable to arbitrary precision, in contrast to the setting of unrelated machines. The previously fastest EPTAS for $Q||C_{\max}$ is due to Jansen, Klein and Verschae [25, 26], and runs in time $2^{\mathcal{O}\left(1/\varepsilon \log^4(1/\varepsilon)\right)} + N^{\mathcal{O}(1)}$. Since their result first appeared in 2016, it has been an open question whether the exponential dependency on $1/\varepsilon$ can be improved. In particular, there is a gap to the best-known lower bound, which shows that a run time of $2^{\mathcal{O}\left((1/\varepsilon)^{1-\delta}\right)} + N^{\mathcal{O}(1)}$ is not possible for any $\delta > 0$, assuming the Exponential-Time Hypothesis [12]. It is unknown, whether this gap can be improved to $\delta = 0$, even under stronger assumptions such as Gap-ETH [33]. Further, a gap remains to the best-known run time $2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon) \log(\log(1/\varepsilon)))} + N^{\mathcal{O}(1)}$ for the case of unit speeds $s_1 = \cdots = s_m$ [8]. Our work shows that this gap could be closed with an algorithm for MILPs with few constraints, as efficient as those available for ILPs with few constraints.

Combinatorial results. Our main combinatorial result improves on the fundamental support size bound of Proposition 1 with regard to the new parameter $A_{\max}$. Prior work used the maximum norm $\Delta := \|A\|_{\max} = \max_{i,j} |A_{i,j}|$, the largest absolute value of an entry in the constraint matrix A. Instead, we use the 1-norm $A_{\max} := \|A\|_1 = \max_i \|A_i\|_1$, the largest 1-norm of any column A_i in A. The matrix 1-norm is sub-multiplicative, consistent with the vector 1-norm and recognizes some sparsity, all in contrast to the maximum norm. This makes $A_{\max}$ a natural parameter to consider for studying the properties of ILPs. We show:

▶ **Theorem 2.** *Any feasible bounded ILP with an m-row constraint matrix A with 1-norm $A_{\max}$ has an optimal solution with support size $s \leq m \cdot (\log(3A_{\max}) + \sqrt{\log(A_{\max})})$.*

We also derive parametric support size bounds in linearized form, like $s \leq 1.1m \cdot (3.42A_{\max})$. As all our upper bounds can equally be derived with $A_{\max}$ replaced by $\sqrt{m}\Delta$, we thereby improve on the constants of Gribanov et al. [19] in this form. For the parameter $A_{\max}$, we show an asymptotically matching lower bound on the support size of an optimal solution:

▶ **Theorem 3.** *For any $m \in \mathbb{Z}_{\geq 0}$ and any $A_{\max} \in \mathbb{Z}_{\geq 1}$, there is an ILP with m constraints, $n := m \cdot (\lfloor \log(A_{\max}) \rfloor + 1) \geq m \cdot \log(A_{\max})$ variables, and 1-norm $A_{\max}$ of the constraint matrix, whose unique optimal solution is the $\mathbf{1}$-vector.*

To obtain this lower bound, we adapt the previously best lower bound of $m \log(\Delta)$ on the support size of an optimal solution by Berndt, Jansen and Klein [9].

Algorithmic results. We use our upper bounds on the support sizes of optimal solutions to ILPs from Theorem 2 as *one* ingredient to obtain new algorithmic results. Namely, we design a new EPTAS for $Q||C_{\max}$, which is asymptotically faster than all previous EPTAS for $Q||C_{\max}$. See Table 1 for a survey of prior work on approximation algorithms for $Q||C_{\max}$.

▶ **Theorem 4.** *There is an algorithm for $Q||C_{\max}$ that, for any $\varepsilon > 0$ and any set of N jobs, computes a $(1+\varepsilon)$-approximate schedule in time $2^{\mathcal{O}\left(1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon))\right)} + \mathcal{O}(N)$.*

Table 1 History of selected complexity results for approximating $Q||C_{\max}$.

authors	year	approximation	run time
Gonzales, Ibarra, Sahni [17]	1977	$2 - \frac{2}{M+1}$	$\mathcal{O}(N \log(N))$
Cho, Sahni [13]	1980	$1 + \sqrt{\frac{M-1}{2}}$	$\mathcal{O}(N \log(N))$
Woeginger [38]	1999	$2 - \frac{1}{M}$	$\mathcal{O}(N \log(N))$
Hochbaum, Shmoys [22]	1988	$1+\varepsilon$	$N^{\mathcal{O}(1/\varepsilon^2 \log(1/\varepsilon))}$
Azar, Epstein [4]	1998	$1+\varepsilon$	$N^{\mathcal{O}(1/\varepsilon^2)}$
Jansen [23]	2010	$1+\varepsilon$	$2^{\mathcal{O}(1/\varepsilon^2 \log^3(1/\varepsilon))} + N^{\mathcal{O}(1)}$
Jansen, Robenek [28]	2012	$1+\varepsilon$	$2^{\mathcal{O}(1/\varepsilon^2 \log^3(1/\varepsilon))} + N^{\mathcal{O}(1)}$
Jansen, Klein, Verschae [25, 26]	2016	$1+\varepsilon$	$2^{\mathcal{O}\left(1/\varepsilon \log^4(1/\varepsilon)\right)} + N^{\mathcal{O}(1)}$
This paper	2023	$1+\varepsilon$	$2^{\mathcal{O}\left(1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon))\right)} + \mathcal{O}(N)$

Compared to previous works, we devise a novel MILP formulation for $Q||C_{\max}$ whose constraint matrix has small column norm. Solving this new formulation not only yields the fastest known EPTAS for $Q||C_{\max}$, but also an EPTAS which is conceptually much simpler than all previous ones. In particular, we introduce the following simplifications in our algorithm compared to the previous EPTAS results:

- In contrast to all other previous algorithms, we do not need a case distinction that splits the algorithm and its analysis into three or four different cases, depending on the processing time ratios and numbers of machines. Our algorithm handles all these cases simultaneously by incurring an increase in the constants hidden within the $\mathcal{O}$ terms.
- We set up an MILP where both jobs and machines with similar size and speed are combined into few distinct job and machine classes. All but the longest jobs and machines are then scheduled fractionally. Rounding the fractional allocations for relatively tiny jobs in a machine class was usually done via a separate algorithm by Lenstra, Shmoys and Tardos [32]. Instead, we pack these jobs with a simple greedy strategy.
- We assign relatively huge jobs of each rounded machine speed using basic linear program properties. This creates a linear instead of logarithmic overhead, which we accommodate by increasing the number of integer variables by a constant factor. Previous EPTAS mostly used a complex algorithm by Jansen [24] for a particular bin packing problem.

The only remaining algorithm used as a black-box is the well-known algorithm by Lenstra [31] for solving MILPs with constant dimension (resp. its improvement due to Kannan [29]). To show the versatility of our approach, we applied it to three related scheduling problems. Due to space constraints, the details of these applications are only in the full version [7].

High-Multiplicity Scheduling. We study the problem $Q|HM|C_{\max}$, where both jobs *and* machines are given in the succinct high-multiplicity encoding. We are not aware of any prior constant-factor approximation for $Q|HM|C_{\max}$, only for the restricted setting where only the jobs are given in a high-multiplicity encoding by Filippi and Romanin-Jacur [16].

▶ **Corollary 5.** *There is an algorithm for $Q|HM|C_{\max}$ that, for any $\varepsilon > 0$ and any instance $\mathcal{I}$, computes a $(1+\varepsilon)$-approximate schedule in time $2^{\mathcal{O}(1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon)))} + \langle\mathcal{I}\rangle^{\mathcal{O}(1)}$.*

Few Different Machine Speeds. In the special case of $Q||C_{\max}$ with only k distinct machine speeds, the run time of our algorithm can be improved.

▶ **Theorem 6.** *There is an algorithm for $Q||C_{\max}$ that, for any $\varepsilon > 0$, any set of N jobs and any k distinct machine speeds, computes a $(1+\varepsilon)$-approximate schedule in time $2^{\mathcal{O}(k \cdot 1/\varepsilon \log(1/\varepsilon) \log(\log(1/\varepsilon)))} + \mathcal{O}(N)$.*

Few Different Uniform Machine Types. Jansen and Maack [27] posed $R_K Q||C_{\max}$, a generalization of $Q||C_{\max}$ where each job can have up to K different processing times.

▶ **Theorem 7.** *There is an algorithm for $R_K Q||C_{\max}$ that, for any $\varepsilon > 0$ and any set of N jobs, computes a $(1+\varepsilon)$-approximate schedule in time $2^{\mathcal{O}(K \log(K) 1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon)))} + \mathcal{O}(K \cdot N)$.*

2 Preliminaries

We use $\log(2) = 1$, i.e., base 2 logarithms and denote Euler's number by $e := \exp(1)$. For a vector $\boldsymbol{v}$, let $v_{\min} := \min_\ell v_\ell$ and $v_{\max} := \max_\ell v_\ell$ be its extremal entries and $\operatorname{supp}(\boldsymbol{v}) := \{\ell \mid \boldsymbol{v}_\ell \neq 0\}$ its *support*, i.e., the set of indices with non-zero entries. For an instance $\mathcal{I}$, its *encoding size* $\langle\mathcal{I}\rangle$ is given by $\langle\mathcal{I}\rangle := \sum_{x \in I}(\log(|x|+1)+1)$, the sum over the sizes in binary representations of all quantities. For example, an instance $\mathcal{I}$ of $Q||C_{\max}$ has size $\langle\mathcal{I}\rangle$ logarithmic in the job processing times and machine speeds, but linear in the number of jobs and machines. We generally use i as index for machines, and j as index for jobs.

Mixed-Integer Linear Programs. The set of feasible solutions of any MILP is

$$\mathcal{Q} \coloneqq \left\{ \boldsymbol{x} \in \mathbb{Z}^n_{\geq 0}, \boldsymbol{y} \in \mathbb{R}^r_{\geq 0} \mid \begin{pmatrix} A & B \\ 0 & C \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \boldsymbol{b} \right\} \tag{MILP}$$

for matrices $A \in \mathbb{Z}^{m \times n}$, $B \in \mathbb{Z}^{m \times r}, C \in \mathbb{Z}^{s \times r}$ and a vector $\boldsymbol{b} \in \mathbb{Z}^{m+s}$. The encoding size $\langle \mathit{MILP} \rangle$ of (MILP) is logarithmic in the absolute value $\Delta \coloneqq \max_{i,j} |A_{i,j}|$ of the largest coefficient and right-hand side $\boldsymbol{b}$, but linear in the number of variables and constraints.

We will make use of the following classical result for finding solutions of (MILP), which was proved first by Lenstra [31] and obtained with improved run time by Kannan [29].

▶ **Proposition 8** (Kannan [29]). *For any instance of* (MILP)*, in time* $2^{\mathcal{O}(n \log(n))} \langle \mathit{MILP} \rangle^{\mathcal{O}(1)}$ *one either finds a solution* $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Q}$ *or determines that* $\mathcal{Q} = \emptyset$*.*

In the following, we reproduce two useful lemmata about the structure of (MILP) solutions, which are inherited from its integral and fractional parts.

▶ **Lemma 9.** *For any instance of* (MILP) *and any* $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \mathcal{Q}$*, in time* $\langle \mathit{MILP} \rangle^{\mathcal{O}(1)}$ *we can find* $(\hat{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \in \mathcal{Q}$ *such that* $\tilde{\boldsymbol{y}}$ *is a vertex solution of the following restricted LP:*

$$\left\{ \boldsymbol{y} \in \mathbb{R}^r_{\geq 0} \mid \begin{pmatrix} B \\ C \end{pmatrix} \cdot \boldsymbol{y} = \boldsymbol{b} - \begin{pmatrix} A \\ 0 \end{pmatrix} \cdot \hat{\boldsymbol{x}} \right\} \ . \tag{R-LP}$$

Proof. By assumption, (R-LP) is feasible, as $\hat{\boldsymbol{y}}$ is a solution. With the ellipsoid algorithm [20, Remark 6.5.2] we find a vertex solution $\tilde{\boldsymbol{y}}$ of (R-LP) in polynomial time. ◀

▶ **Lemma 10.** *For any instance of* (MILP) *and any* $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \mathcal{Q}$ *there is some* $(\tilde{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \mathcal{Q}$ *such that* $\tilde{\boldsymbol{x}}$ *has minimum support* $|\operatorname{supp}(\boldsymbol{x})|$ *of all solutions* $\boldsymbol{x}$ *to the restricted ILP*

$$\left\{ \boldsymbol{x} \in \mathbb{Z}^n_{\geq 0} \mid \begin{pmatrix} A \\ 0 \end{pmatrix} \cdot \boldsymbol{x} = \boldsymbol{b} - \begin{pmatrix} B \\ C \end{pmatrix} \cdot \hat{\boldsymbol{y}} \right\} \ . \tag{R-ILP}$$

Proof. By assumption, (R-ILP) is feasible, as $\hat{\boldsymbol{x}}$ is a solution. Hence, it also has a solution $\tilde{\boldsymbol{x}}$ with minimum support size. Then $(\tilde{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \mathcal{Q}$ holds because of $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \mathcal{Q}$ and $A \cdot \hat{\boldsymbol{x}} = A \cdot \tilde{\boldsymbol{x}}$. ◀

Importantly, Lemma 10 implies that *any* support size bound for an ILP can be directly applied to the integer variables of an MILP. One of these applications is an algorithm to solve MILPs with few constraints. This was presented explicitly and analyzed in terms of m and Δ by Rohwedder and Verschae [36, p. 30], see also Dadush et al. [14]. For us, the crucial underlying idea to efficiently solve MILPs is:

▶ **Lemma 11.** *For any instance of* (MILP) *and any* $s \leq n$*, in time* $2^{\mathcal{O}(s \log(n))} \cdot \langle \mathit{MILP} \rangle^{\mathcal{O}(1)}$ *we either find a solution* $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Q}_{\leq s} \coloneqq \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Q} \colon |\operatorname{supp}(\boldsymbol{x})| \leq s\}$ *of bounded support, or determine that* $\mathcal{Q}_{\leq s} = \emptyset$*.*

Proof. We exhaustively try all choices of $\operatorname{supp}(\boldsymbol{x})$, which are $\binom{n}{s} \leq n^s$ candidates, from $|\operatorname{supp}(\boldsymbol{x})| = 0$ up to s. For each choice we restrict (MILP) to $\operatorname{supp}(\boldsymbol{x})$, by fixing all other integer variables to 0. With Proposition 8 we either find a solution with s integral variables, or determine the infeasibility, in time $s^{\mathcal{O}(s)} \langle \mathit{MILP} \rangle^{\mathcal{O}(1)}$. The run time follows from $s \leq n$. ◀

Note that a support size bound s on any solution of (R-ILP) allows us to use Lemma 11 to find a solution with support size s, or decide the infeasibility of the entire (MILP). Lemma 11 directly extends to optimizing a linear objective function; and to finding a non-zero solution of minimum support, which might be of interest for augmentation algorithms.

3 Refined Support Size Bounds for Integer Linear Programs

In this section, we refine the general support size bounds independent of n for integer linear programs. Previous such bounds used as parameters the number of constraints m, and the largest absolute value Δ of an entry in the constraint matrix A. In our MILP formulation for $Q||C_{\max}$, we bound the support size by the *maximum* 1*-norm* of a column vector, denoted by $A_{\max} := \|A\|_1 = \max_{i=1,\dots,n} \|A_i\|_1$. Clearly, $\Delta \leq A_{\max} \leq m \cdot \Delta$ holds, but it also means that support size bounds using only m and Δ are too coarse for some ranges. In this section, we only consider feasible ILPs $\mathcal{L} \neq \emptyset$ with $\operatorname{rank}(A) = m$. Let $S := \operatorname{supp}(\boldsymbol{v})$ be the support of the vertex $\boldsymbol{v}$, and let $s := |S|$ be the size of the support. Our results are based on:

▶ **Proposition 12** (Aliev et al. [2, Thm. 1(2)]). *Any ILP $\mathcal{L}$ with constraint matrix $A \in \mathbb{Z}^{m\times n}$ has an optimal solution $\boldsymbol{v} \in \mathcal{L}$ with support size $s := |\operatorname{supp}(\boldsymbol{v})| \leq m + \log(\sqrt{\det(A \cdot A^T)})$.*

In this form, the determinant of the support size bound can depend on n. The strength of Proposition 12 is the ability to restrict A to the columns with non-zero variables. For our vertex solution $\boldsymbol{v}$ with support $S = \operatorname{supp}(\boldsymbol{v})$, this is exactly A_S, the columns of the variables in the support. Aliev et al. [2] used the inequality $\sqrt{\det(A_S \cdot A_S^T)} \leq (\sqrt{s}\Delta)^m$ to ultimately obtain the support size bound $s \leq 2m \log(2\sqrt{m}\Delta)$ [3, Thm. 1(ii)]. We analyze the term $\det(A_S \cdot A_S^T)$ in the more fine-grained parameter $A_{\max}$ to obtain a tighter bound:

▶ **Lemma 13.** *For any matrix $A_S \in \mathbb{Z}^{m\times s}$ it holds that $\sqrt{\det(A_S \cdot A_S^T)} \leq (\sqrt{s/m} \cdot A_{\max})^m$.*

Proof. The matrix $G := A_S \cdot A_S^T$ is symmetric and positive semi-definite. If G has an eigenvalue of 0 then $\det(G) = 0$ and the inequality holds. We will thus assume that G is positive definite, i.e., all eigenvalues are positive. It is a classical result that for positive definite matrices, the Hadamard inequality can be strengthened to $\det(G) \leq \prod_{i=1}^m G_{i,i}$, the product of the diagonal entries $G_{i,i}$. We refer to a modern presentation by Browne et al. [10, Thm. 2] for this fact. As $G = A_S \cdot A_S^T$, it is sufficient to bound $\varphi(A_S) := \prod_{i=1}^m \sum_{j=1}^s A_{S;i,j}^2$ subject to $\sum_{i=1}^m |A_{S;i,j}| \leq A_{\max}$ for $j = 1, \dots, s$ by $(s/m \cdot A_{\max}^2)^m$ to obtain our result. We will first characterize a matrix A_S such that $\varphi(A_S)$ is maximal and then relate $\varphi(A_S)$ to $(s/m \cdot A_{\max}^2)^m$. As all entries $A_{S;i,j}$ of A_S occur as squares or absolute values in the optimization, we can assume $A_{S;i,j} \geq 0$ in the following. As φ is monotone in each variable, the matrix A_S that maximizes $\varphi(A_S)$ under the condition $\sum_{i=1}^m |A_{S;i,j}| \leq A_{\max}$ will fulfill these constraints with equality, i.e., $\sum_{i=1}^m |A_{S;i,j}| = A_{\max}$ for $j = 1, \dots, s$. Renaming $A_{S;i,j}$ to $x_{i,j}$ thus gives us the optimization problem:

$$\max \prod_{i=1}^{m} \sum_{j=1}^{s} x_{i,j}^2 \quad \text{s.t.} \sum_{i=1}^{m} x_{i,j} = A_{\max} \quad x_{i,j} \geq 0 \quad \text{for } i = 1, \dots, m; j = 1, \dots, s \ .$$

To bound the optimal solutions to this program, we first consider optimal solutions over the same region with objective function $\sum_{i=1}^m \sum_{j=1}^s x_{i,j}^2$. This is a convex objective function over a polyhedral region. By Bauer's maximum principle [6], the maximum is assumed at a vertex and thus has s non-zero variables. Consequently, we have $\sum_{i=1}^m \sum_{j=1}^s x_{i,j}^2 \leq s \cdot A_{\max}^2$. Now, consider the problem of maximizing $\max \prod_{i=1}^m y_i$ with $\sum_{i=1}^m y_i \leq s \cdot A_{\max}^2$ and $y_i \geq 0$. The logarithm is monotone, so we can apply it to the objective, giving $\sum_{i=1}^m \log(y_i)$ instead. For any solution $x_{i,j}^*$ maximizing $\sum_{i=1}^m \sum_{j=1}^s x_{i,j}^2$, we can compute the y_i^* with $y_i^* = \sum_{j=1}^s {x_{i,j}^*}^2$ that will maximize $\sum_{i=1}^m \log(y_i)$. As the logarithm is concave, we can thus maximize $\sum_{i=1}^m \log(y_i)$ by $y_1^* = \dots = y_m^* = s/m \cdot A_{\max}^2$, as we could otherwise improve a solution by re-balancing it. Hence, $\varphi(A_S) \leq (s/m \cdot A_{\max}^2)^m$, which implies our inequality. ◀

Importantly, by substituting $A_{\max}$ with $\sqrt{m}\Delta$, our inequality in Lemma 13 becomes the one used by Aliev et al. [2]. Therefore, any of the following results can also be obtained for $\sqrt{m}\Delta$ instead of $A_{\max}$. Proposition 12 and Lemma 13 imply the essential intermediate result:

▶ **Corollary 14.** *Any ILP $\mathcal{L}$ with constraint matrix $A \in \mathbb{Z}^{m\times n}$ has an optimal solution $\boldsymbol{v} \in \mathcal{L}$ with $|\operatorname{supp}(\boldsymbol{v})| = s$ such that $s/m \leq 1 + \log(\sqrt{s/m} \cdot A_{\max}) = 1 + \log(A_{\max}) + \log(s/m)/2$.*

The proof of Proposition 12 uses Siegel's Lemma, a deep result from transcendental number theory. In contrast, Berndt et al. [9] derive a support size bound of the same form using only elementary methods, but with worse constants. Building on their approach, we also derive a simple combinatoric, but weaker bound similar to Corollary 14 in the extended version [7].

▶ **Lemma 15.** *Any ILP $\mathcal{L}$ with constraint matrix $A \in \mathbb{Z}^{m\times n}$ has an optimal solution $\boldsymbol{v} \in \mathcal{L}$ with $|\operatorname{supp}(\boldsymbol{v})| = s$ such that $s/m \leq 1 + \log(e) + \log(1 + (s/m) \cdot A_{\max})$.*

Unfortunately, both sides of Corollary 14 are still dependent on s. We resolve Corollary 14 for s in two ways, both parametric in the trade-off between constant and super-constant terms. In the first approach, we bound the logarithm by its tangents.

▶ **Lemma 16.** *For any $\alpha > 0$ and $x > 0$, it holds $\log(x) \leq \alpha \cdot x - \log(e) + \log(\log(e)/\alpha)$.*

Proof. At $x = \log(e)/\alpha$, both sides are $\log(\log(e)/\alpha)$, and the derivatives are α. Hence, the affine function of the right-hand side upper bounds the left-hand side, as $\log(x)$ is concave. ◀

This direct approach allows us to give simple and short formulas for the bounds.

▶ **Theorem 17.** *For any $\alpha \in (0, 1)$ there is an optimal solution $\boldsymbol{v}$ of ILP with*

$$s \leq m \cdot \log(\sqrt{2\log(e)/(e \cdot \alpha)} \cdot A_{\max})/(1 - \alpha) \ .$$

Proof. Applying Lemma 16 with 2α to Corollary 14 yields

$$\begin{aligned} s/m &\leq 1 + \log(A_{\max}) + (2\alpha \cdot s/m - \log(e) + \log(\log(e)/(2\alpha)))/2 \\ &\leq \log(2A_{\max}) + \alpha \cdot s/m + \log\left(\sqrt{\frac{\log(e)}{2\alpha}}\right) \leq \log\left(\sqrt{\frac{2\log(e)}{e\cdot\alpha}} \cdot A_{\max}\right) + \alpha \cdot s/m \ . \end{aligned}$$

We subtract $\alpha \cdot s/m$, and multiply by $m/(1 - \alpha)$, which proves the claim for $0 < \alpha < 1$. ◀

For example, we can set $\alpha = 1/2$ or $\alpha = 1/11$ to obtain the bounds $s \leq 2m \cdot \log(1.46 \cdot A_{\max})$ and $s \leq 1.1m \cdot \log(3.42 \cdot A_{\max})$. This approach, however, only gives bounds with coefficient strictly larger than 1 for the leading term. To reduce this to 1, we make use of advanced analytical function techniques, to tighter analyze the inequality of Corollary 14.

▶ **Lemma 18.** *For $s, m, A_{\max} \geq 1$, the inequality in Corollary 14 is equivalent to*

$$s/m \leq -\log(e) \cdot \mathcal{W}_{-1}(-1/(2\log(e)A_{\max}^2))/2,$$

where $\mathcal{W}_{-1}$ is the -1 branch of the Lambert $\mathcal{W}$-function, the inverse function of $x \mapsto xe^x$.

Proof. We substitute s/m by $-\log(e) \cdot y/2$, and rearrange to obtain:

$$\begin{aligned} & -\log(e) \cdot y/2 \leq 1 + \log(A_{\max}) + \log(-\log(e) \cdot y/2)/2 \\ \Leftrightarrow \quad & \log(-\log(e) \cdot y) + \log(e) \cdot y \geq -(2 + 2\log(A_{\max}) - 1) = -1 - 2\log(A_{\max}) \\ \Leftrightarrow \quad & -\log(e) \cdot y \cdot 2^{\log(e)\cdot y} \geq 1/(2 \cdot A_{\max}^2) \Leftrightarrow y \cdot e^y \leq -1/(2\log(e) \cdot A_{\max}^2) \ . \end{aligned}$$

For the right-hand side $z \coloneqq -1/(2\log(e) \cdot A_{\max}^2)$, we have $-1/e \leq -1/(2\log(e)) \leq z \leq 0$. Therefore, the inequality is satisfied exactly when $\mathcal{W}_{-1}(z) \leq y \leq \mathcal{W}_0(z)$ holds, where $\mathcal{W}_k$ are the real branches of the aforementioned Lambert $\mathcal{W}$-function.

Next, we show $y \leq \mathcal{W}_0(z)$ does not restrict any relevant values. For $x \in [-1/e, 0]$, we have $W_0(x) \geq e \cdot x$. Furthermore $s/m \cdot A_{\max}^2 \geq 1 \geq e/4$ holds for the relevant values of $s/m \geq 1$ and $A_{\max} \geq 1$. Therefore $\mathcal{W}_0(z) \geq -e/(2\log(e)A_{\max}^2) \geq -2 \cdot (s/m)/\log(e) = y$. This shows that the positive solutions are only constrained from above, by $\mathcal{W}_{-1}(z)$. Applying the resubstitution of y to its lower bound gives the claimed result. ◀

Now, we apply techniques analogous to Chatzigeorgiou [11] to prove parametric bounds for the $\mathcal{W}_{-1}(z)$ branch, which we optimize for $z \to 0$ instead of $z \to -1/e$.

▶ **Lemma 19.** *For any $\alpha > 0$ and $u \geq 0$, it holds $-\mathcal{W}_{-1}(-e^{-u-1}) \leq u + \sqrt{2\alpha \cdot u} + \alpha - \ln(\alpha)$.*

Proof. Chatzigeorgiou [11] showed for $x = -\mathcal{W}_{-1}(-e^{-u-1}) - 1$ that $g(x) = u$, where $g(x) \coloneqq x - \ln(1+x)$. To show our result, it thus suffices to show that for all $x \in \mathbb{R} > 0$ and some additive term β, only dependent on α, it holds that

$$-\mathcal{W}_{-1}(-e^{-u-1}) = x + 1 \leq g(x) + \sqrt{2\alpha \cdot g(x)} + \beta + 1 = u + \sqrt{2\alpha \cdot u} + \beta + 1 \ .$$

By definition of g, this reduces to showing $f(x) \coloneqq -\ln(1+x) + \sqrt{2\alpha \cdot (x - \ln(1+x))} + \beta \geq 0$. The function f has a unique minimum, since we will show that $f'(x) = 0$ has only one solution and $\lim_{x\to\infty} f(x) = \infty$. Consider the critical condition:

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) = \frac{-1}{1+x} + \frac{(1 - \frac{1}{1+x}) \cdot \alpha}{\sqrt{2\alpha(x - \ln(1+x))}} = 0 \quad \Leftrightarrow \quad \frac{x\alpha}{\sqrt{2\alpha(x - \ln(1+x))}} = 1 \ . \tag{1}$$

We show that Equation 1 has only one solution, a global minimum, because the second derivative of $f(x)$ is always positive, making $f(x)$ monotonously increasing.

$$\frac{\mathrm{d}}{\mathrm{d}x^2} f(x) = \frac{-x(1 - \frac{1}{1+x})\alpha^2}{(2\alpha(x - \ln(1+x)))^{3/2}} + \frac{\alpha}{\sqrt{2\alpha(x - \ln(1+x))}} > 0 \quad \Leftrightarrow$$

$$\frac{-\alpha^2 x^2/(1+x)}{2\alpha(x - \ln(1+x))} + \alpha > 0 \quad \Leftrightarrow \quad \frac{x^2}{1+x} < 2(x - \ln(1+x)) \quad \Leftrightarrow \quad \frac{x(2+x)}{2(1+x)} \geq \ln(1+x)$$

We know Equation 1 holds at the global minimum. Hence, substituting it in the inequality $f(x) \geq 0$ and applying Lemma 16 on $\ln(1+x)$ bounds the minimal value of $f(x)$ by

$$f(x) = -\ln(1+x) + \alpha x + \beta \geq -\alpha(1+x) - \ln(1/\alpha) + 1 + \alpha x + \beta \geq 0 \quad \Leftrightarrow \quad \beta \geq \alpha + \ln(1/\alpha) - 1 \ .$$

We conclude that $-\mathcal{W}_{-1}(-e^{-u-1}) \leq u + \sqrt{2\alpha \cdot u} + \alpha + \ln(1/\alpha)$, as desired. ◀

From Lemma 18 and Lemma 19 we derive our asymptotically tight support size bound.

▶ **Theorem 20.** *For any $\alpha' > 0$ there is an optimal solution $\boldsymbol{v}$ of ILP with*

$$s \leq m \cdot (\log(A_{\max}) + \sqrt{\alpha'(\log(A_{\max}) + 0.05)} + \alpha'/2 + \log(\sqrt{1/\alpha'}) + 1.03) \ .$$

Proof. We need to rewrite the argument $z \coloneqq -1/(2\log(e)A_{\max}^2)$ in Lemma 18 to the form $-e^{-u-1}$ used in Lemma 19. Hence, solving $z = -e^{-u-1}$ gives $u = \ln(2\log(e)A_{\max}^2/e) = 2\ln(A_{\max}) + \ln(2\log(e)/e)$. We now substitute α by $\alpha'/\log(e)$ to get log instead of ln, and through calculation obtain the bound:

$$
\begin{aligned}
s/m &\leq -\log(e)\mathcal{W}_{-1}(z)/2 \leq \log(e)(u+\sqrt{2\alpha\cdot u}+\alpha-\ln(\alpha))/2\\
&\leq \log(e)(u+\sqrt{2\alpha'/\log(e)\cdot u}+\alpha'/\log(e)-\ln(\alpha'/\log(e)))/2\\
&\leq \log(e)u/2+\sqrt{\log(e)\alpha' u/2}+\alpha'/2-\log(\alpha'/\log(e))/2\\
&\leq \log(A_{\max})+\log(2\log(e)/e)/2+\sqrt{\log(e)\alpha' u/2}-\log(\alpha'/\log(e))/2+\alpha'/2\\
&\leq \log(A_{\max})+\sqrt{\alpha'(\log(A_{\max})+\log(2\log(e)/e)/2)}+\alpha'/2+\log(\log(e)\sqrt{2/(e\alpha')}) \ .
\end{aligned}
$$

Inserting numerical values for terms independent of α' and $A_{\max}$ gives the desired result. ◀

For $\alpha'=1$ and $A_{\max}\geq 1$ we obtain the particularly simple bounds

$$
s \leq m\cdot(\log(A_{\max})+\sqrt{\log(A_{\max})+0.05}+1.53) \leq m\cdot(\log(3A_{\max})+\sqrt{\log(A_{\max})}) \ .
$$

This immediately implies our main support size bound:

▶ **Theorem 2.** *Any feasible bounded ILP with an m-row constraint matrix A with 1-norm $A_{\max}$ has an optimal solution with support size $s\leq m\cdot(\log(3A_{\max})+\sqrt{\log(A_{\max})})$.*

In order to understand how tight our bound is, we adapt a construction by Berndt et al. [9] to obtain an asymptotically matching lower bound on the support size:

▶ **Theorem 3.** *For any $m\in\mathbb{Z}_{\geq 0}$ and any $A_{\max}\in\mathbb{Z}_{\geq 1}$, there is an ILP with m constraints, $n\coloneqq m\cdot(\lfloor\log(A_{\max})\rfloor+1)\geq m\cdot\log(A_{\max})$ variables, and 1-norm $A_{\max}$ of the constraint matrix, whose unique optimal solution is the $\mathbf{1}$-vector.*

Proof. With $d\coloneqq\lfloor\log(A_{\max})\rfloor$ we construct an ILP as follows:

$$
\max \begin{pmatrix} 3^0 & \cdots & 3^d & 3^0 & \cdots & 3^d & \cdots & 3^0 & \cdots & 3^d \end{pmatrix}\cdot\boldsymbol{x} \quad \text{s.t.} \quad \boldsymbol{x}\in\mathbb{Z}_{\geq 0}^{m(d+1)},
$$

$$
\begin{pmatrix} 2^0 & \cdots & 2^d & 0 & & & \cdots & & & 0 \\ 0 & \cdots & 0 & 2^0 & \cdots & 2^d & \cdots & 0 & \cdots & 0 \\ & \vdots & & & & & & \ddots & & \\ 0 & & & \cdots & & & 0 & 2^0 & \cdots & 2^d \end{pmatrix}\cdot\boldsymbol{x} = \begin{pmatrix} 2^{d+1}-1 \\ 2^{d+1}-1 \\ \vdots \\ 2^{d+1}-1 \end{pmatrix} \ .
$$

Because of $\sum_{i=0}^{d}2^i=2^{d+1}-1$, the $\mathbf{1}$-vector is a solution. All coefficients are positive. Hence, in any solution the value of the variables with coefficient 2^d must be less than 2. For any other variable x_i, if it is $x_i\geq 2$, we can increase the objective by setting $x_i\coloneqq x_i-2; x_{i+1}\coloneqq x_{i+1}+1$. Since all objective coefficients are positive, the $\mathbf{1}$-vector is the unique optimal solution. ◀

Hence, Theorem 20 is exact in the dominant term. We pose the question, whether there is a support size bound of the form $m\cdot\log(c\cdot A_{\max})$ for some constant c, as an open problem.

4 An Efficient Approximation Scheme for Makespan Minimization on Uniformly Related machines

Our algorithm follows a typical structure of approximation schemes. First, we *preprocess* the input, by discarding jobs and machines which are so short or slow that assigning them naïvely is acceptable. Next, we perform a *binary search* on the makespan, to reduce the optimization problem to a feasibility problem. Then, we *round* the remaining processing times and machine speeds, according to our makespan guess, to make the resulting instance more structured. Now, we *construct* an MILP, whose feasibility is equivalent to the existence of a schedule. Finally, we solve the MILP and *transform* the solution into a schedule.

4.1 Preprocessing

We reduce the number of parameters bounding the instance to the number of jobs N and a constant fraction δ of the approximation guarantee ε. To enforce $N \geq M$ we potentially drop the $M - N$ slowest machines, as there is an optimal solution not assigning them a job.

Step 1: Removing Negligible Machines and Jobs. We remove all machines slower than $\delta \cdot s_{\max}/N$ and all jobs shorter than $\delta \cdot p_{\max}/N$. Compensating for the lost processing times on a longest job and the machine speeds on a fastest machine introduces an approximation error of at most a factor $(1+\delta)$ each. Now we have $p_{\min} > \delta \cdot p_{\max}/N$ and $s_{\min} > \delta \cdot s_{\max}/N$ and the largest ratios of job processing times $p_{\max}/p_{\min} < N/\delta$ and machine speeds $s_{\max}/s_{\min} < N/\delta$ are bounded only by the parameters N and δ.

Step 2: Preround the Inputs. To achieve a linear run time, we preround the machine speeds and processing times to fewer distinct values. These are rerounded again more carefully at every iteration of the binary search, reducing the run time at the cost of a limited accuracy loss. We round every processing time p_j and machine speed s_i down to the next power of $(1 + \delta)$, introducing errors of no more than $(1 + \delta)$ by construction. Let $\tilde{\eta}_j$ be the number of jobs with rounded processing times $\tilde{p}_j$ be and $\tilde{\mu}_i$ be the number of machines with rounded speed $\tilde{s}_j$. Due to step 1, the amount of *distinct* values after rounding is bounded by $\log_{1+\delta}(p_{\max}/(p_{\max}\delta/N)) = \log_{1+\delta}(s_{\max}/(s_{\max}\delta/N)) \in \mathcal{O}(1/\delta \log(1/\delta \cdot N))$. Because our inputs are sorted, the rounding above can be performed in time $\mathcal{O}(N + 1/\delta \log(1/\delta \cdot N))$.

Step 3: Binary Search for the Makespan. We reduce finding the optimal makespan $\mathrm{OPT}(\mathcal{I})$ to successively checking whether a schedule with makespan T is realizable. The processing time of a longest job on a fastest machine is a lower bound: $\mathrm{OPT}(\mathcal{I}) \geq p_{\max}/s_{\max}$. The schedule assigning all jobs to a fastest machine proves $\mathrm{OPT}(\mathcal{I}) \leq \sum_{j=1}^{N} p_i/s_{\max} \leq N \cdot p_{\max}/s_{\max}$. Hence, we can use a binary search for $\mathrm{OPT}(\mathcal{I})$ in the interval $[p_{\max}/s_{\max}, N \cdot p_{\max}/s_{\max}]$ of ratio N. As accuracy up to a factor $(1+\delta)$ is sufficient, we only need to consider integer powers of $(1 + \delta)$. Our binary search therefore adds a factor of $\mathcal{O}(\log_{1+\delta}(N)) = \mathcal{O}(1/\delta \log(N))$ to the run time of the following steps. We denote the current makespan in the binary search by T. This transforms the problem into either finding a schedule with makespan $(1 + \mathcal{O}(\delta)) \cdot T$, or deciding that no schedule with makespan T exists.

Step 4: Rounding Machine Speeds and Job Processing Times. A $(1 + \varepsilon)$-approximate schedule σ satisfies, for each machine i, the equivalent inequalities

$$\sum_{j \in \sigma^{-1}(i)} \frac{p_j}{s_i} \leq (1 + \varepsilon) \cdot T \quad \Leftrightarrow \quad \sum_{j \in \sigma^{-1}(i)} \frac{p_j}{T} \leq (1 + \varepsilon) \cdot s_i \ . \tag{2}$$

With the aforementioned bounds on the makespan we have enforced the descending chain

$$s_{\max} \geq p_{\max}/T \geq p_{\min}/T > \delta \cdot p_{\max}/(N \cdot T) \geq \delta s_{\max}/N^2 \ .$$

Therefore, all relevant quantities – especially the scaled processing times p_j/T – are in the interval $I \coloneqq (\delta \cdot s_{\max}/N^2, s_{\max}]$, whose left and right end are within a ratio N^2/δ. See also Figure 1 for an overview on the relations between the parameters. We now scale each processing time p_j with $1/T$; this yields an instance with scaled processing times $\tilde{p}_j = 1/T \cdot p_j$, equivalent to our original instance by Equation 2. Our goal is now to cover our interval I by as few subintervals as possible. To this end, we adapt an approach by Berndt

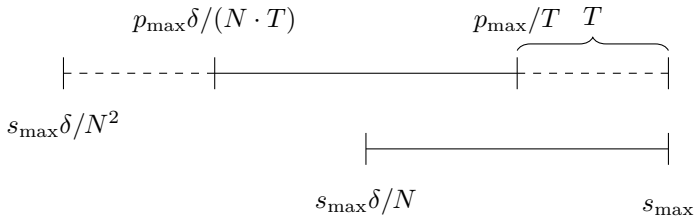


Figure 1 Overview on the range of parameters.

et al. [8] which combines exponential and linear rounding to obtain both sufficient accuracy with few values and useful structural properties. With $\kappa \coloneqq \lceil \log(1/\delta \cdot N^2) \rceil$, consider the points $b_{k,0} \coloneqq s_{\max} \cdot 2^{-k}$ for $k = 0, \dots, \kappa$. The intervals $[b_{k+1,0}, b_{k,0}]$ become exponentially finer, but have a ratio of 2, not the necessary $(1+\delta)$. Therefore, with $\lambda \coloneqq \lceil 1/\delta \rceil$, we add the points $b_{k,\ell} \coloneqq (1 - \ell/(2\lambda)) \cdot b_{k,0}$ for $k = 0, \dots, \kappa - 1$ and $\ell = 0, \dots, \lambda$, which is a linear interpolation between $b_{k,0}$ and $b_{k+1,0}$. Simple calculations show the relations $b_{k,\ell-1}/b_{k,\ell} = 1 + 1/(2\lambda - \ell) \leq 1 + \delta$ and $b_{k+1,0} = b_{k,0}/2 = b_{k,\lceil 1/\delta \rceil}$ for all $k = 0, \dots, \kappa - 1$ and $\ell = 1, \dots, \lambda$, which means we have achieved the necessary precision. Crucially, two jobs within a linear interval of the same parity combine exactly to a job in the next larger linear interval. Formally, this is described by

$$b_{k,\ell} + b_{k,\ell'} = \left(2 - \frac{\ell + \ell'}{2\lambda}\right) \cdot b_{k,0} = \left(1 - \frac{(\ell + \ell')/2}{2\lambda}\right) \cdot b_{k-1,0} = b_{k-1,\frac{\ell+\ell'}{2}} \tag{3}$$

for all $0 \leq \ell \leq \ell' \leq \lambda$ with $\ell + \ell'$ divisible by 2, i.e., ℓ and ℓ' are both odd or both even. Equation 3 will allow us to significantly reduce the number of configurations and the maximum 1-norm of a column. To simplify our notations, we re-index the values of $b_{k,\ell}$ in descending order by setting $b_r \coloneqq b_{k(r),\ell(r)}$ with $k(r) = \lfloor (r-1)/\lambda \rfloor$ and $\ell(r) = (r-1) \bmod \lambda$, such that $b_{k \cdot \lambda + \ell + 1} = b_{k,\ell}$. Therefore, the interval I is covered completely by the $\tau \coloneqq \kappa \cdot \lambda \in \mathcal{O}(1/\delta \log(1/\delta \cdot N))$ intervals $(b_{r+1}, b_r]$ for $r = 1, \dots, \tau$. Finally, we round every machine speed $\tilde{s}_i$ as well as every scaled processing time $\tilde{p}_j$ in $(b_{r+1}, b_r]$ up to b_r. Let μ_i be the number of machines with scaled speed b_i. Let η_j be the number of jobs with scaled processing time b_j. This takes time $\mathcal{O}(1/\delta \log(1/\delta \cdot N))$, which means steps 1 to 4 can be performed in total time $\mathcal{O}(N + 1/\delta \log(N)(1/\delta \log(1/\delta \cdot N) + \tau)) = \mathcal{O}(N + 1/\delta^2 \log^2(1/\delta \cdot N)) \subseteq \mathcal{O}(1/\delta^{2+2\cdot 2}) + \mathcal{O}(N)$.

4.2 Solving an MILP Formulation

Consider the resulting instance after all steps from subsection 4.1 have been applied. Nearly all previous approaches used a mix of *configuration* variables that determine the complete schedule of a machine and *assignment* variables that determine the position of a single job. We combine these different variables into a unified structure called *recursive* configurations. The core idea of our formulation is that an additional machine i of speed b_i can be simulated by placing a *corresponding* job of the same size b_i on a faster machine i' with $b_{i'} < b_i$. In other words, by placing more jobs than the problem requires, we are also allowed to use more machines of the same size than the problem provides. By applying this idea recursively, we can cover a large range of job processing times with configurations of limited range only, as the virtual machines allow us to merge several short jobs (with respect to to a certain machine speed) into a long job. This approach allows us to successively build a configuration from other configurations. Combined with the rounding scheme of Berndt et al. [8] we thereby significantly reduce the overall necessary number of configurations.

For $k = 0, \dots, \kappa$, we define the set $G_k \coloneqq \{r \in \{1, \dots, \tau\} \mid b_r \in [b_{k,0}, b_{k,\lambda}]\}$ of indices of affine slices in our rounding scheme, separated into $G_k^{\text{even}} \coloneqq \{r \in G_k \mid r = 0 \bmod 2\}$ and $G_k^{\text{odd}} \coloneqq G_k \backslash G_k^{\text{even}}$. For $i = 1, \dots, \tau$, we define the set $H_i \coloneqq \{j \in \{1, \dots, \tau\} \mid b_j \in (\delta b_i, b_i]\}$ of indices of *long* job speeds, greater than δb_i and less than the entire machine speed b_i. We will now define *configurations*. All configurations are vectors $\boldsymbol{\gamma}$ with τ entries, each representing multiples of scaled processing times in $\boldsymbol{b} = (b_1, \dots, b_\tau)$. In the following, we describe the configurations for machines with speed b_i. The set $\mathcal{C}_i^{(1)}$ contains all the exact combinations $b_j + b_{j'} = b_i$ as described in Equation 3. With $\boldsymbol{e}_j \in \{0,1\}^\tau$ being the j-th unit vector, let

$$\mathcal{C}_i^{(1)} \coloneqq \{\boldsymbol{e}_j + \boldsymbol{e}_{j'} \mid j, j' \in G_{k(i)-1} \text{ and } j + j' = 2 \cdot (i - \lambda)\} \ .$$

The second set $\mathcal{C}_i^{(2)}$ contains the remaining feasible configurations of long jobs, with at most one job at even and odd positions in an affine slice G_k:

$$\mathcal{C}_i^{(2)} \coloneqq \{\boldsymbol{\gamma} \in \{0,1\}^\tau \mid \boldsymbol{\gamma} \cdot \boldsymbol{b} \leq b_i \text{ and } \operatorname{supp}(\boldsymbol{\gamma}) \subseteq H_i \text{ and for all } k = 0, \dots, \kappa - 1 : \sum_{r \in G_k^{\text{even}}} \gamma_r \leq 1 \text{ and } \sum_{r \in G_k^{\text{odd}}} \gamma_r \leq 1\} \ .$$

Finally, the set of configurations $\mathcal{C}_i$ for machines with speed b_i is defined as $\mathcal{C}_i \coloneqq \mathcal{C}_i^{(1)} \cup \mathcal{C}_i^{(2)}$. The total number of entries in a configuration $\boldsymbol{\gamma}$ is at most $\|\boldsymbol{\gamma}\|_1 \leq 2 \log(2 \cdot 1/\delta)$. Only long jobs from H_i are used, and for each k there is at most one job at an even or odd position in G_k, respectively. We can bound the number of configurations in $\mathcal{C}_i^{(1)}$ by λ^2, and the number of configurations in $\mathcal{C}_i^{(2)}$ by $(\lambda^2)^{2\log(2\cdot 1/\delta)} \in 2^{\mathcal{O}(\log^2(1/\delta))}$, which implies $|\mathcal{C}_i| \in 2^{\mathcal{O}(\log^2(1/\delta))}$.

We require integrality in the variables only for the fastest $L \coloneqq \lambda \lceil \log(1/\delta^3 \log(1/\delta)) \rceil \in \mathcal{O}(1/\delta \log(1/\delta))$ machine speeds. Intuitively, we just need to assign configurations integrally on these machines, as all remaining configurations are very short relative to the machines with the fastest speed. Hence, we can assign them fractionally and round them later on. The overhead from rounding is then scheduled on a fastest machine. The resulting MILP is:

$$\begin{aligned}
\sum_{\boldsymbol{\gamma} \in \mathcal{C}_i} x_{i,\boldsymbol{\gamma}} - \mu_i &\overset{(4)}{=} \sum_{i'=1}^{\tau} \sum_{\boldsymbol{\gamma} \in \mathcal{C}_{i'}} \gamma_i \cdot x_{i',\boldsymbol{\gamma}} - \eta_i \overset{(5)}{\geq} 0 && \text{for } i = 1, \dots, \tau \\
x_{i,\boldsymbol{\gamma}} &\geq 0 && \text{for } i = 1, \dots, \tau;\ \boldsymbol{\gamma} \in \mathcal{C}_i \\
x_{i,\boldsymbol{\gamma}} &\in \mathbb{Z}_{\geq 0} && \text{for } i = 1, \dots, L;\ \boldsymbol{\gamma} \in \mathcal{C}_i.
\end{aligned} \tag{recursive-MILP}$$

Recall that the number of jobs with processing time b_j in configuration $\boldsymbol{\gamma}$ is γ_j, the number of machines with speed b_i is μ_i, and the number of jobs with scaled processing time b_j is η_j. The constraints (4) enforce that the number of additional virtual machines of any speed equals the number of additional corresponding jobs of that same size scheduled somewhere else. The constraints (5) ensure that at least as many jobs of each size are assigned in configurations, as are required by the problem. We show (recursive-MILP) is feasible, up to an approximation factor, for a feasible instance.

▶ **Lemma 21.** *If the original $Q||C_{\max}$-instance $\mathcal{I}$ has a schedule σ with makespan T, then* (recursive-MILP) *is feasible for makespan $(1 + 17\delta) \cdot T$.*

Proof. The schedule σ specifies which jobs are scheduled on any machine i and that those have sufficient speed. We perform steps 1–4, which uses additional speed of at most a factor $(1 + \delta)^7$. If a single job of processing time b_i has been assigned to a machine of speed b_i by σ, then we create a new configuration which assigns this job on that machine, and speed up the

machine by a factor $(1+\delta)$. Otherwise, the difficulty is finding configurations for jobs which are not long, i.e., less than $\delta \cdot b_i$. We repeatedly combine pairs of jobs according to Equation 3 until we have at most one job at an even and odd position in each affine slice G_k on a machine. We partition these jobs j by their value of $\ell = \lfloor \log_\delta(b_j/b_i) \rfloor$, i.e., into slices with ratio δ. The total processing time of jobs in slice ℓ is bounded by $b_i \cdot \delta^\ell \cdot 2 \sum_{j=0}^{\infty} 2^{-j} = 4 \cdot b_i \cdot \delta^\ell$. We iteratively bundle all jobs from the same slice, starting with the last slice until only the first slice is left, into at most 8 configurations of size $b_i \cdot \delta^\ell$. Each configuration can be packed at least half full by the greedy algorithm. The resulting configurations for slice ℓ might have to be rounded in size to the next b_r. Hence, the additional speed introduced for slice ℓ is bounded by $8 \cdot b_i \cdot \delta^\ell \cdot (1+\delta)$. After the creation of such a configuration, if possible, it gets combined again, which uses no additional speed. Eventually, only jobs in the slice for $\ell = 0$ are left. These can be scheduled exactly in a configuration on the machine, by the premise. The additional load incurred on a machine can be bounded by $\sum_{\ell=1}^{\infty} 8 \cdot b_i \cdot \delta^\ell \cdot (1+\delta) = 8 \cdot (1+\delta)/(1-\delta) \cdot \delta \cdot b_i$. For $\delta \leq 1/35$ this is less than $9 \cdot \delta \cdot b_i$ and in total a factor $(1+\delta)^7(1+9\delta) \leq (1+17\delta)$ on the makespan, as claimed. The values $\boldsymbol{x}$ derived from this satisfy constraints (4) and (5) by construction. ◀

For (recursive-MILP), the integer subproblem has $m = 2L \in \mathcal{O}(1/\delta \log(1/\delta))$ constraints and maximal 1-norm of a column $\max \|A_i\|_1 \in \mathcal{O}(\log(1/\delta))$. The first L machine speeds have $L \cdot 2^{\mathcal{O}(\log^2(1/\delta))} = 2^{\mathcal{O}(\log^2(1/\delta))}$ configurations, i.e., integer variables. By Theorem 2 and Lemma 10, if there is a feasible solution of (recursive-MILP), then there is also one with $\mathcal{O}(1/\delta \log(1/\delta) \log(\log(1/\delta)))$ positive integer variables. Lemma 11 thus implies that we can solve (recursive-MILP)(S) in time $2^{\mathcal{O}(1/\delta \log^3(1/\delta) \log(\log(1/\delta)))} \cdot \log(N)^{\mathcal{O}(1)}$, as the encoding size is $\langle (\text{recursive-MILP})(S) \rangle \leq (1/\delta \log(N))^{\mathcal{O}(1)}$. Note that this is the dominant run time in terms of $1/\delta$. We thus either find a feasible solution for the current makespan guess T, or discover that no such solution exists. In the latter case, we discard our current makespan guess and increase it in the next step of the binary search.

4.3 Constructing a Schedule

We need to construct a schedule from a solution $\boldsymbol{x}^\star$ to (recursive-MILP). By constraints (5), we know that $\boldsymbol{x}^\star$ schedules all jobs in some configuration. By constraints (5), we know that $\boldsymbol{x}^\star$ schedules each job in some configuration. By constraints (4), the number of virtual machines equals the additional number of corresponding jobs with equal size scheduled in some configuration. Assigning all configurations to machines within a makespan of approximately T therefore gives a valid schedule, implemented by Algorithm AssignConfsToMachines.

▶ **Lemma 22.** *Algorithm AssignConfsToMachines gives a schedule of makespan at most* $(1+5\delta) \cdot T$ *from a feasible solution* $\boldsymbol{x}^\star$ *to* (recursive-MILP) *in time* $2^{\mathcal{O}(1/\delta)} + \mathcal{O}(N \log^2(N))$.

Proof. The algorithm assigns $\lfloor x_{i,\boldsymbol{\gamma}} \rfloor + 1 \geq x_{i,\boldsymbol{\gamma}}$ configurations, at least as many as $\boldsymbol{x}^\star$ uses. By constraints (5), all jobs get assigned, as they are assigned before the additional jobs corresponding to virtual machines. We always have at least as many virtual machines as $\boldsymbol{x}^\star$, because of the extra configuration added to a fastest machine. We thus need to guarantee that the additional speed scheduled on a fastest machine is sufficiently bounded. The variables of $\boldsymbol{x}^\star$ corresponding to the fastest L machines are already integral. Hence, the first b_i, for which additional speed is put onto the fastest machine, is at most $\delta^3 \cdot s_{\max}/\log(1/\delta)$. There are $\lceil \log(1/\delta) \rceil \lambda + 1$ constraints on the variables $x_{i,\boldsymbol{\gamma}}$ for $\boldsymbol{\gamma} \in \mathcal{C}_i$. Consequently, at most that many variables $x_{i,\boldsymbol{\gamma}}$ can be positive in a vertex solution of the projected LP, which we can find in polynomial time by Lemma 9. As each of these has a size b_i, the total additional speed assigned to a fastest machine is bounded by

AssignConfsToMachines
Input: A feasible solution $\boldsymbol{x}^\star$ to (recursive-MILP).
Output: An approximate schedule σ to the pre-processed instance.

for decreasing machine speeds b_i :
  for each $\boldsymbol{\gamma} \in \mathcal{C}_i$:
    assign $\lfloor x_{i,\boldsymbol{\gamma}} \rfloor$ copies of $\boldsymbol{\gamma}$ to machines with speed b_i
    if $x_{i,\boldsymbol{\gamma}} \notin \mathbb{Z}_{\geq 0}$: assign another copy of $\boldsymbol{\gamma}$ to a fastest machine
    for each processing time b_j used in $\boldsymbol{\gamma}$:
      assign as many jobs of processing time b_j to machines with configuration $\boldsymbol{\gamma}$
      // stop when all jobs are packed or all configurations are filled
      for each job in a configuration $\boldsymbol{\gamma}$ not filled :
        create a virtual machine with the same speed as the corresponding job
return the resulting schedule σ

$$s_{\max}(\lceil \log(1/\delta) \rceil \lambda + 1) \sum_{r=L}^{\infty} b_r = s_{\max}(\lceil \log(1/\delta) \rceil \lambda + 1) \sum_{k=L/\lambda}^{\infty} \sum_{\ell=0}^{\lambda-1} (2 - \ell/\lambda) 2^{-k}$$
$$\leq \frac{\delta^3 \cdot s_{\max}}{\log(1/\delta)} (\lceil \log(1/\delta) \rceil \lambda + 1)(1 + 3\lambda) < 5\delta s_{\max} \ .$$

The last inequality holds for $\delta \leq 1/35$. Adding this much speed to a fastest machine results in a schedule with makespan at most $(1 + 5\delta) \cdot T$. The run time needed to construct the schedule is bounded by the number of machines times the effort per machine, resulting in

$$\mathcal{O}(N \cdot \tau) = \mathcal{O}(1/\delta \cdot N \log(1/\delta \cdot N^2)) \subseteq 2^{\mathcal{O}(1/\delta)} + \mathcal{O}(N \log^2(N)) \ . \quad \blacktriangleleft$$

5 Faster Schedule Construction

The results of the previous section already give us an EPTAS for $Q||C_{\max}$ with almost linear run time of $2^{\mathcal{O}(1/\varepsilon \log^3(1/\varepsilon) \log(\log(1/\varepsilon)))} + \mathcal{O}(N \log^2(N))$. Interestingly, the bottleneck (with respect to N) of this approach is the transformation from a valid MILP solution into a feasible schedule. In this section, we give a more conventional $Q||C_{\max}$ MILP formulation (hybrid-MILP) using both configuration and assignment variables to improve the run time in N. First, we give an algorithm to transform a solution of (recursive-MILP) into a solution of (hybrid-MILP) in sublinear run time in N. Then, we show how to construct a schedule from a solution to (hybrid-MILP) in linear time. This allows us to transform a solution of (recursive-MILP) into a valid schedule in linear run time in N.

Note that Lemma 21 constructs a solution to (recursive-MILP) from a schedule to $Q||C_{\max}$. Hence, both formulations are equivalent up to a multiplicative error of $1 + \mathcal{O}(\varepsilon)$.

5.1 The Hybrid-MILP Formulation

Let $\mathcal{C}'_i := \{\gamma \in \mathbb{N}^\tau \mid \boldsymbol{\gamma} \cdot \boldsymbol{b} \leq b_i \text{ and } \operatorname{supp}(\boldsymbol{\gamma}) \subseteq H_i\}$ be the set of configurations of long jobs with range $1/\delta$ for machine i. For any machine speed b_i and corresponding configuration $\gamma \in \mathcal{C}'_i$, let $\operatorname{free}(i, \gamma) := b_i - \boldsymbol{\gamma} \cdot \boldsymbol{b}$ be the speed of the machine that is free after placing the jobs specified by γ. Then (hybrid-MILP) is given by

$$\sum_{\boldsymbol{\gamma}\in\mathcal{C}'_i} x_{i,\boldsymbol{\gamma}} = \mu_i \qquad \text{for } i = 1,\dots,\tau \tag{6}$$

$$\sum_{i=1}^{\tau}\sum_{\boldsymbol{\gamma}\in\mathcal{C}'_i} \gamma_j \cdot x_{i,\boldsymbol{\gamma}} + \sum_{i=1}^{\tau} y_{i,j} = \eta_j \qquad \text{for } j = 1,\dots,\tau \tag{7}$$

$$\sum_{\boldsymbol{\gamma}\in\mathcal{C}'_i} \text{free}(i,\boldsymbol{\gamma}) \cdot x_{i,\boldsymbol{\gamma}} - \sum_{j=1}^{\tau} b_j \cdot y_{i,j} \geq 0 \qquad \text{for } i = 1,\dots,\tau \tag{8}$$

$$\begin{aligned} x_{i,\boldsymbol{\gamma}},\ y_{i,j} &\geq 0 && \text{for } i = 1,\dots,\tau\,; j = 1,\dots,\tau\,; \gamma \in \mathcal{C}_i \\ y_{i,j} &= 0 && \text{for } i = 1,\dots,\tau\,; j \in \{\min(H_i),\dots,\tau\} \\ x_{i,\boldsymbol{\gamma}} &\in \mathbb{Z}_{\geq 0} && \text{for } i = 1,\dots,L\,; \gamma \in \mathcal{C}_i\ . \end{aligned}$$

(hybrid-MILP)

In this formulation, there are no recursive configurations. Instead, we use configuration variables $x_{i,\gamma}$, indicating how often a configuration γ is used on machine i. Short jobs, taking up speed less than δb_i on machine i, are handled via assignment variables $y_{i,j}$ indicating how many jobs of size b_j are assigned to machines of speed b_i. The constraints (6) enforce that every machine is assigned a configuration, the constraints (7) guarantee that every job is scheduled somewhere, and the constraints (8) make sure that the speed used by short jobs is at most the speed left free by configurations.

We now convert a solution $\boldsymbol{x}^\star$ of (recursive-MILP) into a solution $(\boldsymbol{x}, \boldsymbol{y})$ of (hybrid-MILP).

ConvertMILPsolution
Input: A feasible solution $\boldsymbol{x}^\star$ to (recursive-MILP).
Output: A feasible solution $\boldsymbol{x}, \boldsymbol{y}$ to (hybrid-MILP).

for $i = 1,\dots,\tau$:
  initialize $\eta'_i := \eta_i$ and $x_{i,\gamma} = x^\star_{i,\gamma}$ for $\gamma \in \mathcal{C}_i$, otherwise $x_{i,\gamma} = 0$
for decreasing machine speeds b_i :
  decrease arbitrary $x_{i,\gamma} > 0$ until $\sum_{\gamma\in\mathcal{C}'_i} x_{i,\gamma} = \mu_i$
  do count the number of jobs ζ_j in configurations $x_{i,\gamma}$ for $j \in H_i$
    for every job size b_j with $\zeta_j \geq \eta'_j$:
      substitute $\zeta_j - \eta'_j$ many jobs in appropriate $x_{i,\gamma}$ with $x_{j,\gamma'}$
      (that is, decrease $x_{j,\gamma'}, x_{i,\gamma}$ and increase $x_{i,\gamma''}$, where γ'' is γ with
        jobs j replaced with γ' and jobs shorter than δb_i in γ' dropped)
  until there are no more $\zeta_j \geq \eta'_j$
  decrease every η'_i by ζ_j
for increasing machine speeds b_i :
  calculate the free machine speed $z_i := \sum_{\gamma\in\mathcal{C}'_i} \text{free}(i,\gamma) \cdot x_{i,\gamma}$
  starting with the shortest jobs b_j, increase $y_{i,j}$, decrease z_i and η'_j
  until z_i is 0, $\eta'_j = 0$ or $j \in H_i$
return $(\boldsymbol{x}, \boldsymbol{y})$

▶ **Lemma 23.** *Algorithm ConvertMILPsolution converts a solution $\boldsymbol{x}^\star$ of* (recursive-MILP) *into a solution $(\boldsymbol{x}, \boldsymbol{y})$ of* (hybrid-MILP) *in time $2^{\mathcal{O}(1/\delta \log^2(1/\delta))} \log^{\mathcal{O}(1)}(N)$.*

Proof. The algorithm guarantees that the sum of the configuration variables $x_{i,\gamma}$ for machines with speed b_i is exactly μ_i. Hence, constraints (6) are satisfied. Jobs are only replaced once all original jobs have been accommodated and, by constraints (4), there are exactly as many additional virtual machines, as there are additional jobs. Therefore, this step never reduces the total number of configurations for any machine speed below μ_i. After the loop of lines 3-11 on machines with speed b_i, only jobs of size $b_j \in (\delta b_i, b_i]$, or in other words $j \in H_i$, are assigned via configurations. Additional short jobs would have been assigned via the recursive configurations on these machines, which we neglect by stopping at δb_i. Thus, these machines have sufficient speed to handle these short jobs, which would have been assigned to them. Instead of their original order, we assign the short jobs by increasing size. This does not change the total load assigned, which therefore still remains sufficient. As the sorting ensures that any short job assigned is smaller or equal to some short job that would have been assigned by the original order, we do not assign jobs which are no longer tiny. Due to having sufficient machine speed and not dropping jobs anywhere in the algorithm, we assign all real jobs and hence satisfy constraints (7). Finally, constraints (8) are satisfied by construction, as we never overfill the available speed. The number of configurations $|\mathcal{C}_i'|$ is bounded by $2^{\mathcal{O}(1/\delta \log^2(1/\delta))}$ and all other quantities are bounded by $\mathcal{O}(1/\delta \log(1/\delta \cdot N))$. That allows us to analyze the run time of the algorithm to be within the claimed complexity. ◀

5.2 Constructing a Schedule

Now, we have an assignment for all small jobs, albeit with fractional variables. However, we can assign the small jobs integrally faster than if we had to resolve recursive configurations.

▶ **Lemma 24.** *Given a feasible solution of* (hybrid-MILP)*, a schedule with makespan at most $(1 + 9\delta)T$ can be constructed in time $2^{\mathcal{O}(1/\delta \log^2(1/\delta))} + \mathcal{O}(N)$.*

Proof. For the configuration variables, we pursue the same strategy as in Lemma 22, that is, rounding them down and assigning one configuration to a fastest machine for every rounded variable. Through the use of basic solutions, we construct a schedule introducing a multiplicative error of at most$(1 + 5\delta)$ in comparison to the optimal makespan. The process takes time $2^{\mathcal{O}(1/\delta \log^2(1/\delta))} \log^{\mathcal{O}(1)}(N)$ with our increased number of configurations.

For the assignment variables of the short jobs, we first note that any machine speed b_i is assigned at most 2 fractional short jobs, as a variable only becomes fractional when the preceding or current group runs out of speed. As in Lemma 23, we first sort the assignment variables, in time $\mathcal{O}((1/\delta \log(1/\delta \cdot N))^4)$. By increasing every machine speed by a factor of $(1 + 2\delta)$, those two fractional jobs can be placed on an arbitrary machine of speed b_i, without exceeding the speed. We get another overhead of a factor of $(1 + \delta)$ by greedily packing the assigned jobs to machines, overpacking each machine just slightly. This greedy packing takes time $\mathcal{O}(N + (1/\delta \log(1/\delta \cdot N))^2)$. In total the approximation error is bounded by $(1 + 5\delta)(1 + 2\delta)(1 + \delta) \leq (1 + 9\delta)$ for $\delta < 1/35$, as claimed. ◀

By Lemma 21 for an instance $\mathcal{I}$ with makespan $\mathrm{OPT}(\mathcal{I})$ the recursive-MILP with makespan $(1 + 17\delta) \cdot \mathrm{OPT}(\mathcal{I})$ is feasible. We then converted a solution to one of hybrid-MILP with Lemma 23. A solution for hybrid-MILP with makespan $(1 + 17\delta) \cdot \mathrm{OPT}(\mathcal{I})$ then gives a schedule with makespan $(1 + 9\delta)(1 + 17\delta) \cdot \mathrm{OPT}(\mathcal{I})$ by Lemma 24. Hence, for $\varepsilon \leq 1$ we can pick $\delta = \varepsilon/35$ and obtain a schedule with makespan $(1 + \varepsilon)\,\mathrm{OPT}(\mathcal{I})$. All of the above steps take linear run time in N. Therefore we have achieved Theorem 4.

References

1 Iskander Aliev, Gennadiy Averkov, Jesús A. De Loera, and Timm Oertel. Sparse representation of vectors in lattices and semigroups. *Math. Program.*, 192(1-2, Ser. B):519–546, 2022. doi:10.1007/s10107-021-01657-8.

2 Iskander Aliev, Jesús A. De Loera, Friedrich Eisenbrand, T. Oertel, and Robert Weismantel. The support of integer optimal solutions. *SIAM J. Optim.*, 28(3):2152–2157, 2018. doi:10.1137/17M1162792.

3 Iskander Aliev, Jesús A. De Loera, Timm Oertel, and Christopher O'Neill. Sparse solutions of linear Diophantine equations. *SIAM J. Appl. Algebra Geom.*, 1(1):239–253, 2017. doi:10.1137/16M1083876.

4 Yossi Azar and Leah Epstein. Approximation schemes for covering and scheduling in related machines. *Proc. APPROX 1998*, 1444:39–47, 1998. doi:10.1007/BFb0053962.

5 Nikhil Bansal, Tim Oosterwijk, Tjark Vredeveld, and Ruben van der Zwaan. Approximating vector scheduling: almost matching upper and lower bounds. *Algorithmica*, 76(4):1077–1096, 2016. doi:10.1007/s00453-016-0116-0.

6 Heinz Bauer. Minimalstellen von Funktionen und Extremalpunkte. II. *Arch. Math.*, 11:200–205, 1960. doi:10.1007/BF01236933.

7 Sebastian Berndt, Hauke Brinkop, Klaus Jansen, Matthias Mnich, and Tobias Stamm. New support size bounds for integer programming, applied to makespan minimization on uniformly related machines, 2023. arXiv:2305.08432.

8 Sebastian Berndt, Max A. Deppert, Klaus Jansen, and Lars Rohwedder. Load balancing: The long road from theory to practice. In *Proc. ALENEX 2022*, pages 104–116, 2022. doi:10.1137/1.9781611977042.9.

9 Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. New bounds for the vertices of the integer hull. *Proc. SODA 2021*, pages 25–36, 2021. doi:10.1137/1.9781611976496.3.

10 Patrick Browne, Ronan Egan, Fintan Hegarty, and Padraig Ó Catháin. A survey of the Hadamard maximal determinant problem. *Electron. J. Combin.*, 28(4):Paper No. 4.41,35, 2021. doi:10.37236/10367.

11 Ioannis Chatzigeorgiou. Bounds on the Lambert function and their application to the outage analysis of user cooperation. *IEEE Comm. Lett.*, 17(8):1505–1508, 2013. doi:10.1109/LCOMM.2013.070113.130972.

12 Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *J. Comput. Syst. Sci.*, 96:1–32, 2018. doi:10.1016/j.jcss.2018.03.005.

13 Yookun Cho and Sartaj Sahni. Bounds for list schedules on uniform processors. *SIAM J. Comput.*, 9(1):91–103, 1980. doi:10.1137/0209007.

14 Daniel Dadush, Arthur Léonard, Lars Rohwedder, and José Verschae. Optimizing low dimensional functions over the integers. In *Integer Programming and Combinatorial Optimization*, pages 115–126, 2023. doi:10.1007/978-3-031-32726-1_9.

15 Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006. doi:10.1016/j.orl.2005.09.008.

16 Carlo Filippi and Giorgio Romanin-Jacur. Exact and approximate algorithms for high-multiplicity parallel machine scheduling. *J. Sched.*, 12(5):529–541, 2009. doi:10.1007/s10951-009-0122-z.

17 Teofilo Gonzalez, Oscar H. Ibarra, and Sartaj Sahni. Bounds for LPT schedules on uniform processors. *SIAM J. Comput.*, 6(1):155–166, 1977. doi:10.1137/0206013.

18 Ronald L. Graham, Eugene L. Lawler, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, 5:287–326, 1979. doi:10.1016/S0167-5060(08)70356-X.

19 Dmitry Gribanov, Ivan Shumilov, Dmitry Malyshev, and Panos Pardalos. On Δ-modular integer linear problems in the canonical form and equivalent problems. *J. Glob. Optim.*, pages 1–61, 2022. doi:10.1007/s10898-022-01165-9.

20 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2 of *Algorithms and Combinatorics: Study and Research Texts*. Springer Berlin, Heidelberg, 1988. doi:10.1007/978-3-642-97881-4.

21 Christoph Haase and Georg Zetzsche. Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests. *Proc. LICS 2019*, pages 1–14, 2019. doi:10.1109/LICS.2019.8785850.

22 Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, 1988. doi:10.1137/0217033.

23 Klaus Jansen. An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables. *SIAM J. Discrete Math.*, 24(2):457–485, 2010. doi:10.1137/090749451.

24 Klaus Jansen. A fast approximation scheme for the multiple knapsack problem. In *Proc. SOFSEM 2012*, volume 7147 of *Lecture Notes Comput. Sci.*, pages 313–324, 2012. doi:10.1007/978-3-642-27660-6_26.

25 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. In *Proc. ICALP 2016*, volume 55 of *Leibniz Int. Proc. Informatics*, pages Art. No. 72,13, 2016. doi:10.4230/LIPIcs.ICALP.2016.72.

26 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Math. Oper. Res.*, 45(4):1371–1392, 2020. doi:10.1287/moor.2019.1036.

27 Klaus Jansen and Marten Maack. An EPTAS for scheduling on unrelated machines of few different types. *Algorithmica*, 81(10):4134–4164, 2019. doi:10.1007/s00453-019-00581-w.

28 Klaus Jansen and Christina Robenek. Scheduling jobs on identical and uniform processors revisited. In *Proc. WAOA 2011*, volume 7164 of *Lecture Notes Comput. Sci.*, pages 109–122, 2012. doi:10.1007/978-3-642-29116-6_10.

29 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.

30 Viktor Kuncak and Martin Rinard. Towards efficient satisfiability checking for Boolean algebra with Presburger arithmetic. In *Proc. CADE 2021*, volume 4603 of *Lecture Notes Comput. Sci.*, pages 215–230, 2007. doi:10.1007/978-3-540-73595-3_15.

31 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.

32 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Programming*, 46(3, (Ser. A)):259–271, 1990. doi:10.1007/BF01585745.

33 Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense CSPs. *Proc. ICALP 2017*, 80:Art. No. 78,15, 2017. doi:10.4230/LIPIcs.ICALP.2017.78.

34 Timm Oertel, Joseph Paat, and Robert Weismantel. Sparsity of integer solutions in the average case. *Proc. IPCO 2019*, 11480:341–353, 2019. doi:10.1007/978-3-030-17953-3_26.

35 Ian Pratt-Hartmann. On the computational complexity of the numerically definite syllogistic and related logics. *Bull. Symbolic Logic*, 14(1):1–28, 2008. doi:10.2178/bsl/1208358842.

36 Lars Rohwedder. *Algorithms for Integer Programming and Allocation*. phdthesis, Universität Kiel, 2019. URL: https://macau.uni-kiel.de/receive/diss_mods_00026125.

37 Thomas Rothvoss. Integer optimization and lattices, 2016. Lecture Notes. URL: https://sites.math.washington.edu/~rothvoss/lecturenotes/IntOpt-and-Lattices.pdf.

38 Gerhard J. Woeginger. A comment on scheduling on uniform machines under chain-type precedence constraints. *Oper. Res. Lett.*, 26(3):107–109, 2000. doi:10.1016/S0167-6377(99)00076-0.

Improved Guarantees for the A Priori TSP

Jannis Blauth
Research Inst. for Discrete Mathematics, Hausdorff Center for Math., University of Bonn, Germany

Meike Neuwohner
Research Inst. for Discrete Mathematics, Hausdorff Center for Math., University of Bonn, Germany

Luise Puhlmann
Research Inst. for Discrete Mathematics, Hausdorff Center for Math., University of Bonn, Germany

Jens Vygen
Research Inst. for Discrete Mathematics, Hausdorff Center for Math., University of Bonn, Germany

Abstract

We revisit the A PRIORI TSP (with independent activation) and prove stronger approximation guarantees than were previously known. In the A PRIORI TSP, we are given a metric space (V, c) and an activation probability $p(v)$ for each customer $v \in V$. We ask for a TSP tour T for V that minimizes the expected length after cutting T short by skipping the inactive customers.

All known approximation algorithms select a nonempty subset S of the customers and construct a *master route solution*, consisting of a TSP tour for S and two edges connecting every customer $v \in V \setminus S$ to a nearest customer in S.

We address the following questions. If we randomly sample the subset S, what should be the sampling probabilities? How much worse than the optimum can the best master route solution be? The answers to these questions (we provide almost matching lower and upper bounds) lead to improved approximation guarantees: less than 3.1 with randomized sampling, and less than 5.9 with a deterministic polynomial-time algorithm.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases A priori TSP, random sampling, stochastic combinatorial optimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.14

Related Version *Full Version*: https://arxiv.org/abs/2309.10663

Supplementary Material *Software*: https://doi.org/10.60507/FK2/JCUIRI

1 Introduction

Many algorithms for stochastic discrete optimization problems sample a sub-instance, solve the resulting deterministic problem (often by some approximation algorithm), and extend this solution to the original instance [8, 11, 14, 15, 16, 25]. A nice and well-studied example is the A PRIORI TRAVELING SALESPERSON PROBLEM (A PRIORI TSP), which is the focus of this paper. What guarantee can we obtain by such an approach, even if we take an optimal sample? If we sample randomly, according to which distribution? What guarantee can we obtain by a deterministic polynomial-time algorithm? These are the questions addressed in this paper.

In the A PRIORI TSP (with independent activation), we are given a (semi-)metric space (V, c); the elements of V are called *customers*. Each customer v comes with an *activation probability* $0 < p(v) \leq 1$, so it will be active independently with probablity $p(v)$. However, we need to design a TSP tour T (visiting all of V) *before* knowing which customers will be active. *After* we know which customers are active we can cut the tour T short by skipping the inactive customers. The goal is to minimize the expected cost of the resulting tour (visiting the active customers).

Note that computing an optimum a priori tour is APX-hard as the metric TSP is APX-hard [23], which is the special case where all activation probabilities are 1. We study approximation algorithms. A *ρ-approximation algorithm* for the A PRIORI TSP is a polynomial-time algorithm that computes a tour of expected cost at most $\rho \cdot \mathrm{OPT}$ for any given instance, where OPT denotes the expected cost of an optimum a priori tour.

Shmoys and Talwar [25] devised a randomized 4-approximation algorithm and a deterministic 8-approximation algorithm. A randomized constant-factor approximation algorithm was discovered independently by Garg, Gupta, Leonardi and Sankowski [11]. The randomized Shmoys–Talwar algorithm easily improves to a 3.5-approximation by using the Christofides–Serdyukov algorithm instead of the double tree algorithm as a subroutine for TSP (as noted by [7]), and slightly better using the new Karlin–Klein–Oveis Gharan algorithm [20]. The deterministic algorithm was improved to a 6.5-approximation by van Zuylen [28]; a slight improvement of this guarantee follows from the recent deterministic version of the Karlin–Klein–Oveis Gharan algorithm [21].

All known approximation algorithms for the A PRIORI TSP are of the following type. Select a nonempty subset S of customers and find a TSP tour for S (the *master tour*). Connect each other customer $v \in V \setminus S$ with a pair of parallel edges to a nearest point $\mu(v)$ in the master tour. We call this a *master route solution*. Once we know the set of active customers, we pay for the entire master tour (pretending to visit also its inactive customers!) and pay $2c(\mu(v), v)$ for each active customer v outside S to cover the round trip visiting v from $\mu(v)$. See Figure 1 for an example. Of course, we could cut the resulting tour shorter (we visit some inactive customers, and we visit some customers several times), but we will not account for this possible gain (unless fewer than two customers are active).

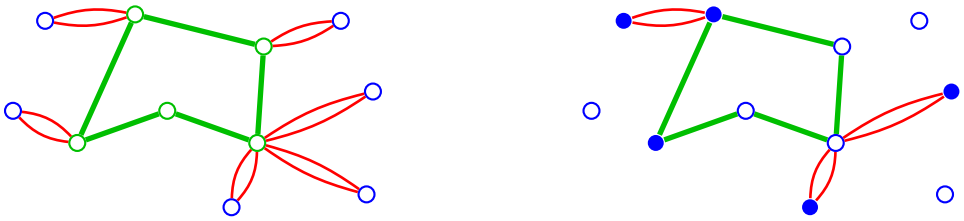


Figure 1 Left: A master route solution with a master tour (green, thick) and connections of the other customers to that master tour (red, curved). Right: After knowing which customers are active (filled), the master route solution reduces to a tour visiting all of the master tour and the other active customers.

1.1 Motivating questions

We start by reviewing the randomized algorithm by Shmoys and Talwar [25]. If fewer than two customers are active, any a priori tour can be cut short to a single point, resulting in cost zero. The algorithm by Shmoys and Talwar [25] selects each customer v independently into S with probability $p(v)$: exactly the activation probability. Assuming that the resulting set S is nonempty, there exists an associated master route solution with expected cost at most

$$\mathrm{MR}(S) \;:=\; \mathbb{E}_{A\sim p}\left[\mathbb{1}_{|A|\geq 2} \cdot \left(\mathrm{OPT}_{\mathrm{TSP}}(S,c) + 2\cdot \sum_{v\in A} c(v,S)\right)\right].$$

Here $\mathrm{OPT}_{\mathrm{TSP}}(S,c)$ denotes the length of an optimum TSP tour for S, and $c(v,S) = \min\{c(v,s) : s \in S\}$ denotes the distance between v and a nearest customer in S (which is zero if $v \in S$); moreover, $\mathbb{E}_{A\sim p}$ denotes the expectation when the set A of active customers is

sampled with respect to the given activation probabilities. Later on, $\mathbb{P}_{A\sim p}$ is used analogously. We multiply with $\mathbb{1}_{|A|\geq 2}$ (which is 1 if $|A| \geq 2$ and 0 otherwise) because the cost of the solution is zero if fewer than two customers are active.

If there is a customer d with $p(d) = 1$ (a *depot*), then S is never empty and we can bound

$$\mathrm{MR}(S) \;\leq\; \mathrm{OPT}_{\mathrm{TSP}}(S,c) + 2\cdot\mathbb{E}_{A\sim p}\left[\sum_{v\in A\setminus\{d\}} c(v, S\setminus\{v\})\right].$$

Note that the above upper bound also accounts for connecting active customers in S to the nearest other customer in S, which is not necessary but will allow the following. Taking the expectation over the random choice of S, an upper bound on the expected cost of that master route solution is

$$\mathbb{E}_{S\sim p}\left[\mathrm{MR}(S)\right] \;\leq\; \mathbb{E}_{S\sim p}\left[\mathrm{OPT}_{\mathrm{TSP}}(S,c)\right] + 2\cdot\mathbb{E}_{S\sim p}\left[\sum_{v\in S\setminus\{d\}} c(v, S\setminus\{v\})\right]$$

as the probability distributions to choose S and A are identical and the vertices are sampled independently. Since $\sum_{v\in S\setminus\{d\}} c(v, S\setminus\{v\}) \leq \mathrm{OPT}_{\mathrm{TSP}}(S,c)$ for all S and $\mathbb{E}_{S\sim p}\left[\mathrm{OPT}_{\mathrm{TSP}}(S,c)\right] \leq \mathrm{OPT}$, where OPT again denotes the expected cost of an optimum a priori tour, this yields

$$\mathbb{E}_{S\sim p}\left[\mathrm{MR}(S)\right] \;\leq\; 3\cdot\mathrm{OPT}. \tag{1}$$

The work of Shmoys and Talwar [25] implies that (1) also holds when there is no depot and when we take the conditional expectation under the condition that $|S| \geq 2$ (see also [28]). The Shmoys–Talwar algorithm cannot find an optimum TSP tour for S but uses the double tree algorithm with approximation guarantee 2. As noted by [7], one can as well use the Christofides–Serdyukov algorithm with approximation guarantee $\frac{3}{2}$, or in fact any α-approximation algorithm for TSP. Then the expected cost of the resulting master route solution is at most $(\alpha+2)\cdot\mathrm{OPT}$.

This motivates the following questions:

- **(i)** Is it optimal to sample S with exactly the activation probabilities (which is crucially used in the above analysis), or can we improve on the factor $\alpha+2$ by sampling fewer or more?
- **(ii)** How bad can the best master route solution be? We will call this the *master route ratio*: by the Shmoys–Talwar analysis, it is at most 3.
- **(iii)** Can we obtain an approximation guarantee equal to the master route ratio by a master route solution based on random sampling, assuming that we can find optimum TSP tours? What is the best we can achieve with a $\frac{3}{2}$-approximation algorithm for TSP?
- **(iv)** Can we obtain a better deterministic algorithm without a better TSP algorithm?

We give almost complete answers to all these questions.

1.2 Our results

The possibility that we sample the empty set or that no customer is active causes significant complications. The previous works [25] and [28] gave ad hoc proofs that *their* algorithms (which are also formulated with a depot) generalize to the non-depot case. We aim for a general reduction, losing only an arbitrarily small constant: Fortunately, instances in which the expected number of active customers is small can be solved easily with an approximation factor $3+\varepsilon$ (for any $\varepsilon > 0$; similar to [8]), and hence much better than the known guarantees. For instances with a large expected number of active customers, one can assume without

loss of generality (with an arbitrarily small loss) that there is a customer d that is always active, i.e., $p(d) = 1$ (see full version of this paper [4]). So we assume this henceforth and call d the depot. We summarize (and refer to the full version [4] for the proof):

▶ **Theorem 1.** *Let $\varepsilon > 0$ and $\rho \geq 3$ be constants. If there exists a (randomized) polynomial-time ρ-approximation algorithm for instances (V, c, p) of the* A PRIORI TSP *that have a depot (i.e., a customer d with $p(d) = 1$), then there is a (randomized) polynomial-time $(\rho + \varepsilon)$-approximation algorithm for general instances of the* A PRIORI TSP.

The Shmoys-Talwar algorithm [25] includes a customer v into S with probability $p(v)$: the sampling probability is exactly the activation probability. Although this is natural and allows for the simple analysis in Section 1.1 (assuming a depot), we show that this is not optimal. Decreasing the probability of including a customer into the master tour improves the approximation guarantee. To be more precise, in Section 2, we analyze the following *sampling algorithm* for A PRIORI TSP instances with depot. Let $f\colon (0,1] \to [0,1]$ with $f(1) = 1$.

(i) Sample a subset $S \subseteq V$ by including every customer v independently with probability $f(p(v))$.

(ii) Call an α-approximation algorithm for (metric) TSP in order to compute a TSP tour for S, which serves as master tour.

(iii) Connect every customer outside S to the nearest customer in S by a pair of parallel edges.

For a given instance this algorithm has expected approximation ratio at most

$$\frac{1}{\mathrm{OPT}} \cdot \mathbb{E}_{S \sim f \circ p}\left[\alpha \cdot \mathrm{OPT}_{\mathrm{TSP}}(S, c) + 2 \cdot \sum_{v \in V} p(v) \cdot c(v, S)\right] \tag{2}$$

(where $\frac{0}{0} \coloneqq 1$). Shmoys and Talwar [25] used the identity function $f(p) = p$. It is easy to construct examples where sampling less or more is better. For example, if $c(v, w) = 1$ for all $v, w \in V$ with $v \neq w$ (and all activation probabilities except for the depot are tiny), it is best to include only the depot in the master tour: this yields an approximation ratio of 2 instead of 3. On the other hand, if $V = \{v_0, \ldots, v_{n-1}\}$ and $c(v_i, v_j) = \min\{j - i, n + i - j\}$ for $i < j$ (i.e., (V, c) is the metric closure of a cycle), the more we sample, the better. However, even if we choose f depending on the instance, there is a limit on what we can achieve:

▶ **Theorem 2.** *No matter how f is chosen, even depending on the instance in an arbitrary way, the sampling algorithm has no better approximation ratio than*

- 2.655 *even if it computes an optimum TSP tour on the sampled customers;*
- 3.049 *assuming that we never compute a TSP tour on the sampled customers of cost less than* 1.4999 *times the cost of an optimum tour.*

See the full version of our paper [4] for the proof. We do not have a matching upper bound, but we come close. For $\alpha = 1.5$ we prove (in Section 2):

▶ **Theorem 3.** *For $\alpha = 1.5$ and $f(p) = 1 - (1 - p)^{\sigma}$ with $\sigma = 0.663$, the sampling algorithm for* A PRIORI TSP *instances with depot has approximation guarantee less than 3.1.*

Figure 2 shows this function f. Together with Theorem 1 this immediately implies one of our main results:

▶ **Corollary 4.** *There is a randomized* 3.1*-approximation algorithm for* A PRIORI TSP. ⌟

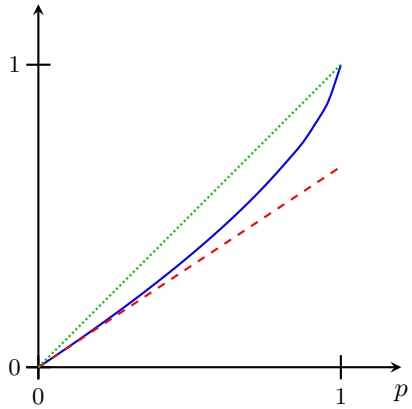


Figure 2 The function $p \mapsto 1 - (1-p)^\sigma$ with $\sigma = 0.663$ (blue, solid) defines the sampling probability in Theorem 3, which is always at most the identity function (green, dotted), and for small p approximately equal to $p \mapsto \sigma \cdot p$ (red, dashed).

We conjecture that the bounds in Theorem 2 are actually attained by the sampling algorithm with $f(p) = 1 - (1-p)^\sigma$, independent of the instance, where σ is a positive constant that depends on α only. See Comment 17 for details.

Having explored the limits of the random sampling approach, one might ask what is the limit of choosing an *optimal* master route solution. By van Zuylen's work [28], the answer to this question is the key to obtain a better deterministic approximation algorithm. Let us define:

▶ **Definition 5** (master route ratio). *The* master route ratio *is defined to be the supremum of*

$$\frac{\min\{\mathrm{MR}(S) : \emptyset \neq S \subseteq V\}}{\mathrm{OPT}}$$

taken over all A PRIORI *TSP instances (where* $\frac{0}{0} := 1$*).*

It is very easy to see that the master route ratio is at least 2 (for example, if $c(v,w) = 1$ for all $v, w \in V$ with $v \neq w$). By the Shmoys–Talwar analysis, it is at most 3. We show in the full version of our paper [4]:

▶ **Theorem 6.** *The master route ratio for* A PRIORI *TSP instances with depot is at least* $\frac{1}{1-e^{-1/2}} > 2.541$ *and less than* 2.6*.*

We conjecture that the master route ratio is exactly $\frac{1}{1-e^{-1/2}}$.

As van Zuylen's [28] analysis reveals (cf. [4]), her algorithm is a $(2+\alpha\rho)$-approximation algorithm if the master route ratio is ρ and we have an algorithm for TSP that guarantees to produce a tour of cost at most α times the value of the subtour relaxation. So our new upper bound on the master route ratio immediately implies a better guarantee (combining Theorems 1 and 6 with $\alpha = \frac{3}{2}$ [27]):

▶ **Corollary 7.** *There is a deterministic 5.9-approximation algorithm for* A PRIORI *TSP.* ⌟

1.3 Our techniques

The lower bounds (Theorem 2 and the lower bound in Theorem 6) are obtained by analyzing simple examples. The main technical difficulty is in proving the upper bounds.

To prove Theorem 3 and the upper bound in Theorem 6, we will show that it suffices to consider instances in which all customers (except the depot) have the same tiny activation probability. We call these instances *normalized*.

▶ **Definition 8.** *Let $\varepsilon > 0$. An instance (V, c, p) of* A PRIORI TSP *is called* ε-normalized *if the instance contains a depot $d \in V$ (with $p(d) = 1$), and $p(v) = \varepsilon$ for all $v \in V \setminus \{d\}$.*

Given an instance of A PRIORI TSP with a depot d, one can transform it to a normalized instance by replacing each customer $v \in V \setminus \{d\}$ by many copies, each with the same tiny activation probability, such that the probability that at least one of these copies is active is roughly $p(v)$. This way, the master route ratio and the approximation guarantee of the sampling algorithm can only get worse. More precisely, we show in the full version of this paper [4]:

▶ **Lemma 9.** *Let $(\varepsilon_i)_{i\in\mathbb{N}} \in (0,1]^{\mathbb{N}}$ with $\lim_{i\to\infty} \varepsilon_i = 0$. Let $\mathcal{I}$ be the class of all ε-normalized instances with $\varepsilon = \varepsilon_i$ for some $i \in \mathbb{N}$. Then*

- **(i)** *The master route ratio is the same when restricting it to instances in $\mathcal{I}$ and when restricting it to all instances with depot.*
- **(ii)** *Let $\sigma \in (0,1)$. Every upper bound on (2) for $f(p) = \sigma p \ \forall p \in (0,1)$ for all instances in $\mathcal{I}$ implies the same upper bound on (2) for $f(p) = 1 - (1-p)^\sigma$ for arbitrary instances with depot.*

On a high level, our proofs of Theorem 3 and Theorem 6 are similar. In both cases we will design a linear program that encodes the metric c by variables and minimizes the expected cost of an optimum a priori tour subject to (a relaxation of) the constraint that the expected cost of the output of the sampling algorithm is at least 1 (for Theorem 3) or the expected cost of any master route solution is at least 1 (for Theorem 6), respectively. Then the reciprocals of the LP values yield the desired upper bounds.

However, this approach has to overcome several obstacles. First, it is not obvious how to encode the metric c by finitely many variables, given that we need to consider arbitrary instance sizes. We do this by fixing an optimum a priori tour T^* (a cyclic order of the customers) and carefully aggregating distances of customer pairs with the same number of hops in between on T^*. Of course we exploit the structure of normalized instances.

In the end, we will (almost) ignore variables that correspond to a very large number of hops (where it is very unlikely that none of the customers "in between" is active). These variables have negligible impact because the probability that these edges occur decreases exponentially with increasing number of hops on T^*, whereas the average length of these edges can only grow linearly due to the triangle inequality.

The next idea is to consider certain structured solutions only. Rather than connecting a customer v that is not in the master tour to the *nearest* customer $\mu(v)$ in the master tour, we consider only two possible members of the master tour: we traverse T^* from v in each of the two possible directions, and consider the first customer that we meet and that is contained in our master tour. None of these two may be a nearest one in the master tour, but we still obtain an upper bound. For bounding the master route ratio, we will in addition only consider master tours whose customers are equidistantly distributed on T^* (except for the depot).

In this way, we obtain an optimization problem for a fixed uniform activation probability p (i.e., for p-normalized instances). However, we must let $p \to 0$ according to Lemma 9 and hence need a description that is independent of p. This is another major obstacle. To overcome it, we use a second level of aggregation (buckets, rounding the number of hops to integer multiples of, say, $\frac{1}{100p}$). However, this causes several difficulties. In the case of the

sampling algorithm, describing the expected cost of the output of the sampling algorithm in terms of the buckets is nontrivial. In case of the master route ratio, the same holds for master route solutions and actually requires a third level of aggregation (bucket intervals).

In the end, we obtain (in both cases) a single, relatively compact, linear program that yields an upper bound for all instance sizes and all activation probabilities from a sequence that converges to zero. We solve the dual LP numerically and just need to check feasibility to prove the desired upper bounds.

1.4 Further related work

The TSP has also been studied under the aspect of robust optimization, where the set of customers that need to be visited is known in advance, but the edge lengths are chosen probabilistically or even adversarially [10, 26]. The a priori optimization problem where the set of customers is chosen adversarially is known as universal TSP [11, 13, 24]. The probabilistic version that we consider was introduced by Jaillet [17] and Bertsimas [2]. Since then, various aspects of the problem have been investigated, including the asymptotic behavior of random instances [2, 3, 5, 17, 18], online variants [11], or exact algorithms [1]. Approximation algorithms have also been studied for general probability distributions [6, 13, 24].

Other problems that have been considered in an a priori setting include vehicle routing, traveling repairman, Steiner tree, and network design [7, 8, 9, 11, 14, 15, 16, 22]. However, none of these works managed to determine the approximation guarantee of their algorithms exactly.

Previous approaches to design a linear program that yields the approximation ratio of a certain algorithm for some optimization problem (e.g., [12, 19]) typically required an infinite family of linear programs and could not obtain a bound for general instances by just solving a single linear program.

2 Upper bound on the approximation ratio of random sampling

In this section we will prove Theorem 3. As mentioned earlier, we will design a single linear program such that the reciprocal of its optimum value is an upper bound on the approximation ratio of the sampling algorithm for a certain class of normalized instances. For this sake, let $\beta, b_0 > 0$ be constants that we will choose later. We will consider ε-normalized instances where ε is of the form $\varepsilon = \frac{\beta}{b}$ for some odd integer $b \geq b_0$. The meaning of these constants will become clear in Section 2.2. For such instances we will obtain an upper bound on the approximation ratio of the sampling algorithm, when sampling each customer with probability σp for $\sigma = 0.663$ (in addition to the depot). Combined with Lemma 9, this immediately yields the same upper bound on the approximation guarantee of the sampling algorithm that samples each customer v with probability $1 - (1 - p(v))^\sigma$ for arbitrary A PRIORI TSP instances with depot.

2.1 An optimization problem to bound the approximation ratio

In this section, we first describe an upper bound for all p-normalized instances (for a fixed uniform activation probability p) by a single optimization problem. We will consider the algorithm that samples each customer with probability σp. Let T^* be a fixed optimum a priori tour, with customers appearing in the order $v_0, v_1, \dots, v_{n-1}$; here v_0 denotes the depot. Let $v_i := v_0$ for $i < 0$ or $i > n - 1$. For $k \in \mathbb{Z}_{\geq 1}$ we define

$$C_k \;:=\; p^2 \cdot \sum_{j \in \mathbb{Z}} c(v_j, v_{j+k}).$$

Observe that only finitely many summands are nonzero. See Figure 3 for an example. Since c is a metric, the numbers C_k are nonnegative and satisfy the triangle inequality, that is, for all $i, j \geq 1$

$$C_{i+j} \;\leq\; C_i + C_j. \tag{3}$$

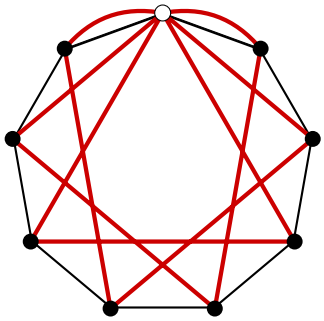


Figure 3 The depot v_0 is the white circle at the top; the tour T^* is drawn in black. Adding up the costs of the edges marked in red gives $\frac{C_3}{p^2}$.

Moreover, we can express the expected cost of T^* in terms of the C_i.

▶ **Proposition 10.** *The expected cost of T^* is exactly*

$$\sum_{i=1}^{\infty}(1-p)^{i-1} \cdot C_i. \tag{4}$$

Proof. Let $1 \leq i \leq n-2$ and $1 \leq j \leq n-i-1$. Then v_j and v_{j+i} are consecutive active customers with probability $p^2 \cdot (1-p)^{i-1}$; note that the cost of the edge $\{v_j, v_{j+i}\}$ is counted with exactly the same coefficient in (4). Moreover, for $1 \leq j \leq n-1$, v_j is the first active customer after the depot with probability $p \cdot (1-p)^{j-1}$, and the cost of the edge $\{v_0, v_j\}$ is counted $\sum_{i=j}^{\infty} p^2 \cdot (1-p)^{i-1} = p \cdot (1-p)^{j-1}$ times in (4). By symmetry, the terms also match for the last active customer before the depot. ◀

We now consider the master route solution resulting from sampling each customer with probability σp (in addition to the depot). Let α again denote the approximation guarantee of the TSP algorithm that we use. We will now show that the expected cost of this master route solution is at most

$$\sigma^2 \sum_{k=1}^{\infty}(1-\sigma p)^{k-1} \cdot \left(\alpha \cdot C_k + 2p \cdot \sum_{i=1}^{k-1} \min\left\{C_i,\, C_{k-i}\right\}\right). \tag{5}$$

By the same argumentation as in the proof of Proposition 10, the master tour has expected cost at most

$$\alpha \cdot \mathbb{E}_{S\sim q}[c(T^*[S])] \;=\; \alpha \cdot \sigma^2 \cdot \sum_{k=1}^{\infty}(1-\sigma p)^{k-1} \cdot C_k,$$

where $q(v) = \sigma p$ for all $v \in V \setminus \{d\}$ and $q(d) = 1$.

Next we bound the expected cost of connecting the active customers to the master tour. Instead of connecting v to the nearest customer in the master tour, we consider only two options: the first sampled customer that we meet when traversing T^* from v in either

direction. Note that sampling v_0 with probability 1 is equivalent to sampling each v_j with $j \leq 0$ and $j \geq n$ with probability σp. Now, for $j \in \mathbb{Z}$ and $k \geq 2$, the probability that v_j and v_{j+k} are sampled, but none of the intermediate customers is, equals $(\sigma p)^2 \cdot (1-\sigma p)^{k-1}$. In this case, the total expected cost of connecting the intermediate active customers can be bounded by $2p \cdot \sum_{i=1}^{k-1} \min\{c(v_j, v_{j+i}), c(v_{j+i}, v_{j+k})\}$. Thus we can bound the expected cost of connecting all active customers to the master tour by

$$\begin{aligned}
&\sum_{k=2}^{\infty} \sigma^2 p^2 \cdot (1-\sigma p)^{k-1} \cdot \sum_{j \in \mathbb{Z}} 2p \cdot \sum_{i=1}^{k-1} \cdot \min\left\{c(v_j, v_{j+i}),\, c(v_{j+i}, v_{j+k})\right\} \\
&\leq\ 2\sigma^2 p^3 \cdot \sum_{k=1}^{\infty} (1-\sigma p)^{k-1} \cdot \sum_{i=1}^{k-1} \min\left\{\sum_{j \in \mathbb{Z}} c(v_j, v_{j+i}),\, \sum_{j \in \mathbb{Z}} c(v_{j+i}, v_{j+k})\right\} \\
&=\ 2\sigma^2 p^3 \cdot \sum_{k=1}^{\infty} (1-\sigma p)^{k-1} \cdot \sum_{i=1}^{k-1} \min\left\{\sum_{j \in \mathbb{Z}} c(v_j, v_{j+i}),\, \sum_{j \in \mathbb{Z}} c(v_j, v_{j+(k-i)})\right\} \\
&=\ 2\sigma^2 p \cdot \sum_{k=1}^{\infty} (1-\sigma p)^{k-1} \cdot \sum_{i=1}^{k-1} \min\left\{C_i,\, C_{k-i}\right\}.
\end{aligned}$$

We conclude that the ratio of (5) to (4) is an upper bound on the approximation guarantee of the sampling algorithm for that instance. Note that the number of customers appears neither in (4) nor in (5). In other words, minimizing (4) subject to the constraints that (5) is equal to 1 and the C_i are nonnegative and satisfy the triangle inequality (3) yields the reciprocal of an upper bound on the approximation guarantee of the sampling algorithm on all p-normalized instances. We arrive at the following optimization problem:

$$\min \sum_{i=1}^{\infty} (1-p)^{i-1} \cdot C_i \qquad \text{(Sampling-OP)}$$

$$\begin{aligned}
\text{subject to} \qquad C_i &\geq 0 && \text{for } i \in \mathbb{N} && (6)\\
C_i + C_j &\geq C_{i+j} && \text{for } i, j \in \mathbb{N} && (7)\\
\sum_{k=1}^{\infty} (1-\sigma p)^{k-1} \cdot \left(\alpha \cdot C_k + 2p \cdot \sum_{i=1}^{k-1} \min\left\{C_i,\, C_{k-i}\right\}\right) &\geq \sigma^{-2}. && && (8)
\end{aligned}$$

Note that in (8) we only require that (5) is at least 1 instead of exactly 1. This does not change the infimum because we can always scale all the C_i's. We have proved:

▶ **Lemma 11.** *Let $0 < p < 1$. The reciprocal of the value of* (Sampling-OP) *is an upper bound on the approximation guarantee for the sampling algorithm with $f(p) = \sigma p$ for all p-normalized instances.* ⌟

2.2 Obtaining a single linear program

Note that we have an infinite set of optimization problems (one for each choice of p), and, in view of Lemma 9, we have to consider the limit for $p \to 0$.

In the following, we require that p is of the form $p = \frac{\beta}{b}$ for some odd integer $b \geq b_0$. Note that $p \to 0$ as $b \to \infty$. In order to obtain a single optimization problem for all such values of p, we put subsequent C_i's into buckets of size b. More precisely, we define buckets

$$B_i \ :=\ \sum_{j=\max\{1, ib-\frac{b-1}{2}\}}^{ib+\frac{b-1}{2}} C_j \qquad (9)$$

for $i \geq 0$. In the following, we show that we can use the constraints in (Sampling-OP) to generate (slightly relaxed) constraints that only depend on these buckets. First, we note that the buckets are chosen such that they still satisfy the triangle inequality.

▶ **Proposition 12.** *For all $i, j \geq 1$,*

$$B_{i+j} \;\leq\; B_i + B_j.$$

Proof. Indeed, using (7), as illustrated in Figure 4,

$$\begin{aligned} B_{i+j} \;&=\; \sum_{k=-\frac{b-1}{2}}^{\frac{b-1}{2}} C_{(i+j)b+k} \;=\; \sum_{k=0}^{\frac{b-1}{2}} C_{(i+j)b-\frac{b-1}{2}+2k} + \sum_{k=1}^{\frac{b-1}{2}} C_{(i+j)b-\frac{b-1}{2}+2k-1} \\ &\leq \sum_{k=0}^{\frac{b-1}{2}} \left(C_{ib-\frac{b-1}{2}+k} + C_{jb+k}\right) + \sum_{k=1}^{\frac{b-1}{2}} \left(C_{ib+k} + C_{jb-\frac{b-1}{2}+k-1}\right) \\ &= \sum_{k=-\frac{b-1}{2}}^{\frac{b-1}{2}} C_{ib+k} + \sum_{k=-\frac{b-1}{2}}^{\frac{b-1}{2}} C_{jb+k} \;=\; B_i + B_j. \end{aligned}$$

◀

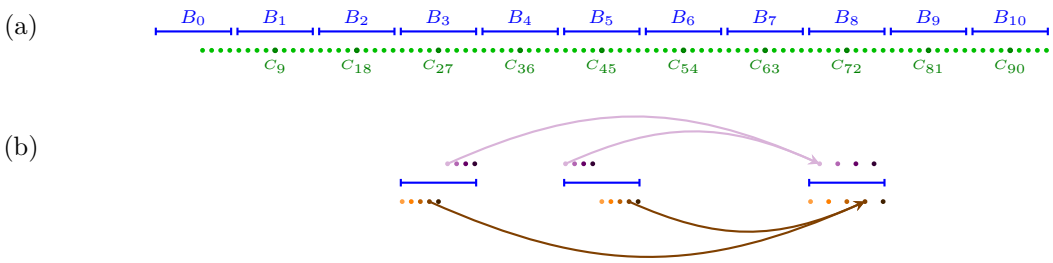


Figure 4 (a): The green dots stand for $C_1, C_2, \ldots$, and the centers of the buckets (C_{ib} for $i \geq 1$) are highlighted. Here the bucket size is $b = 9$, and the blue intervals show the buckets $B_0, B_1, B_2, \ldots$ (b): Combining the triangle inequalities for the C_i's leads to triangle inequalities for the B_i's; here shown for $B_i = B_3$ and $B_j = B_5$: We add up all triangle inequalities for C_k from B_3 and C_ℓ from B_5 where C_k and C_ℓ have the same color; illustrated with $C_{26} + C_{48} \leq C_{74}$ and $C_{28} + C_{41} \leq C_{69}$.

Next we aim for an upper bound on the left-hand side of (8) that only depends on the buckets. First we show:

▶ **Lemma 13.**

$$\sum_{k=1}^{\infty} (1-\sigma p)^{k-1} \cdot \sum_{i=1}^{k-1} \min\left\{C_i,\, C_{k-i}\right\} \;\leq\; b \cdot \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} e^{-(i+j-1)\cdot\sigma b p} \cdot \min\{B_i, B_j\}.$$

Proof. For $i \in \mathbb{Z}_{\geq 0}$, let $I_i = \{\max\{1, ib - \frac{b-1}{2}\}, \ldots, ib + \frac{b-1}{2}\}$ be the set of indices in the i-th bucket. Then

$$\begin{aligned} &\sum_{k=1}^{\infty} (1-\sigma p)^{k-1} \cdot \sum_{i=1}^{k-1} \min\left\{C_i,\, C_{k-i}\right\} \\ &= \sum_{k=1}^{\infty} \sum_{\ell=1}^{\infty} (1-\sigma p)^{k+\ell-1} \cdot \min\left\{C_k,\, C_\ell\right\} \end{aligned}$$

$$\leq \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}(1-\sigma p)^{\max\{0,i+j-1\}\cdot b}\cdot\sum_{k\in I_i}\sum_{\ell\in I_j}\min\{C_k,C_\ell\}$$
$$\leq \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}e^{-(i+j-1)\cdot\sigma bp}\cdot\sum_{k\in I_i}\sum_{\ell\in I_j}\min\{C_k,C_\ell\}.$$

In the last inequality we used $1-x\leq e^{-x}$ for all $x\in\mathbb{R}$. Now, for $i,j\in\mathbb{Z}_{\geq 0}$, consider the complete bipartite graph H where one bipartition consists of $|I_j|$ copies of every element of I_i, and the other bipartition consists of $|I_i|$ copies of every element of I_j. Then

$$\sum_{(k,\ell)\in E(H)}\min\{C_k,C_\ell\} \;=\; |I_i|\cdot|I_j|\cdot\sum_{k\in I_i}\sum_{\ell\in I_j}\min\{C_k,C_\ell\}.$$

We can partition $E(H)$ into $t\coloneqq|I_i|\cdot|I_j|$ perfect matchings $M_1,\dots,M_t$. Then

$$\begin{aligned}\sum_{(k,\ell)\in E(H)}\min\{C_k,C_\ell\} &= \sum_{s=1}^{t}\sum_{(k,\ell)\in M_s}\min\{C_k,C_\ell\}\\ &\leq \sum_{s=1}^{t}\min\left\{\sum_{(k,\ell)\in M_s}C_k,\sum_{(k,\ell)\in M_s}C_\ell\right\}\\ &= \sum_{s=1}^{t}\min\left\{|I_j|\cdot\sum_{k\in I_i}C_k,|I_i|\cdot\sum_{\ell\in I_j}C_\ell\right\}\\ &= |I_i|\cdot|I_j|\cdot\min\left\{|I_j|\cdot B_i,|I_i|\cdot B_j\right\}.\end{aligned}$$

Note that the second equality follows from the fact that $V(H)$ contains $|I_j|$ copies of each element in I_i and vice versa. Moreover, summing over the endpoints of the edges in a perfect matching in M is the same as summing over $V(H)$. Division by $|I_i|\cdot|I_j|$ yields

$$\sum_{k\in I_i}\sum_{\ell\in I_j}\min\{C_k,C_\ell\} \;\leq\; \min\left\{|I_j|\cdot B_i,|I_i|\cdot B_j\right\} \;\leq\; b\cdot\min\{B_i,B_j\}.$$ ◀

Using Lemma 13 and $\beta=bp$, the left-hand side of (8) can be upper bounded by

$$\begin{aligned}&\alpha\cdot\sum_{k=1}^{\infty}(1-\sigma p)^{k-1}\cdot C_k \;+\; 2bp\cdot\sum_{i=0}^{\infty}\sum_{j=0}^{\infty}e^{-(i+j-1)\cdot\sigma bp}\cdot\min\{B_i,B_j\}\\ &\leq\; \alpha\cdot\sum_{k=0}^{\infty}(1-\sigma p)^{\max\{0,kb-\frac{b-1}{2}-1\}}\cdot B_k \;+\; 2bp\cdot\sum_{i=0}^{\infty}\sum_{j=0}^{\infty}e^{-(i+j-1)\cdot\sigma bp}\cdot\min\{B_i,B_j\}\\ &\leq\; \alpha\cdot\sum_{k=0}^{\infty}e^{-(k-1)\cdot\sigma\beta}\cdot B_k \;+\; 2\beta\cdot\sum_{i=0}^{\infty}\sum_{j=0}^{\infty}e^{-(i+j-1)\cdot\sigma\beta}\cdot\min\{B_i,B_j\}.\end{aligned}\tag{10}$$

The last inequality follows from $(1-\sigma p)^{kb-\frac{b-1}{2}-1}\leq(1-\sigma p)^{kb-b}$ and $1+x\leq e^x$ for all $x\in\mathbb{R}$. Note that we still sum over infinitely many variables. Hence, in order to get a finite linear program, we aim for an upper bound on (10) that only depends on the buckets B_i with $i\leq N$ for some integer N that we will choose later. For this, we use the triangle inequality (Proposition 12) to bound the terms depending on buckets B_i with $i>N$ by some term depending on $B_1,\dots,B_N$ only. For large N this will result in a negligible error as the coefficients in (10) decrease exponentially. In Section 2.4 we prove the following bound on the error term:

▶ **Lemma 14.** *Let* $\delta_1 \coloneqq \frac{4\beta}{e^{N\sigma\beta}(e^{\sigma\beta}-1)}$ *and* $\delta_2 \coloneqq \left(\alpha + \frac{2\beta}{e^{N\sigma\beta}(e^{\sigma\beta}-1)}\right) \cdot \frac{e^{-N\sigma\beta}}{(1-e^{-\sigma\beta})^2} \cdot (1+N-e^{-\sigma\beta}N)$. *Then*

$$\alpha \cdot \sum_{k=N+1}^{\infty} e^{-(k-1)\cdot\sigma\beta} \cdot B_k \;+\; 2\beta \cdot \sum_{i,j\in\mathbb{Z}_{\geq 0}:\max\{i,j\}>N} e^{-(i+j-1)\cdot\sigma\beta} \cdot \min\{B_i, B_j\}$$
$$\leq\; \delta_1 \cdot \sum_{k=0}^{N} e^{-(k-1)\cdot\sigma\beta} \cdot B_i + \delta_2 \cdot B_1.$$

Therefore, we get a lower bound on (Sampling-OP) by minimizing $\sum_{i=1}^{\infty}(1-p)^{i-1} \cdot C_i$ subject to (9) and $B_i \geq 0$ for $i \geq 0$, $B_{i+j} \leq B_i + B_j$ for $i, j \geq 1$ with $i + j \leq N$, and

$$(\alpha+\delta_1) \cdot \sum_{k=0}^{N} e^{-(k-1)\cdot\sigma\beta} \cdot B_k + 4\beta \cdot \sum_{j=0}^{N}\sum_{i=0}^{j} e^{-(i+j-1)\cdot\sigma\beta} \cdot \min\{B_i, B_j\} + \delta_2 \cdot B_1 \geq \sigma^{-2}. \quad (11)$$

Note that the objective still contains infinitely many variables and depends on p. The first problem can easily be resolved by bounding

$$\sum_{i=1}^{\infty}(1-p)^{i-1} \cdot C_i \;\geq\; \sum_{i=0}^{\infty}(1-p)^{bi+\frac{b-1}{2}-1} \cdot B_i \;\geq\; \sum_{i=0}^{N}(1-p)^{(i+\frac{1}{2})b} \cdot B_i. \quad (12)$$

It remains to get rid of the dependence on b and p (recall that $p = \frac{\beta}{b}$). To this end, we exploit that $\lim_{b\to\infty}(1-\frac{\beta}{b})^{(i+\frac{1}{2})b} = e^{-(i+\frac{1}{2})\beta}$ for all $i = 0, \ldots, N$, and that by Lemma 9, we can choose b_0 arbitrarily large. This will allow us to conclude that we can replace the objective by $\sum_{i=0}^{N} e^{-(i+\frac{1}{2})\beta} \cdot B_i$ and still obtain an upper bound (see the proof of Lemma 15 for the technical details). Putting everything together, we arrive at the following LP.

$$\min \sum_{i=0}^{N} e^{-(i+\frac{1}{2})\beta} \cdot B_i \quad \text{(Sampling-LP)}$$

$$\text{subject to} \quad (\alpha+\delta_1) \cdot \sum_{k=0}^{N} e^{-(k-1)\cdot\sigma\beta} \cdot B_k + \delta_2 \cdot B_1 + 4\beta \cdot \sum_{j=0}^{N}\sum_{i=0}^{j} e^{-(i+j-1)\cdot\sigma\beta} \cdot M_{i,j} \;\geq\; \sigma^{-2} \quad (13)$$

$$B_i + B_j \;\geq\; B_{i+j} \qquad \text{for } 1 \leq i \leq j \leq N \text{ with } i+j \leq N \quad (14)$$

$$B_i \;\geq\; M_{i,j} \qquad \text{for } 0 \leq i \leq j \leq N \quad (15)$$

$$B_j \;\geq\; M_{i,j} \qquad \text{for } 0 \leq i \leq j \leq N \quad (16)$$

$$B, M \;\geq\; 0. \quad (17)$$

Recall that δ_1 and δ_2 were defined in Lemma 14. We conclude:

▶ **Lemma 15.** *Let* N *be an integer and* $\beta > 0$. *The reciprocal of the optimum value of* (Sampling-LP) *is an upper bound on the approximation guarantee of the sampling algorithm for* $f(p) = 1-(1-p)^{\sigma}$ *(using an* α*-approximation algorithm for TSP), for all* A PRIORI *TSP instances with depot.*

Proof. We compare the value of (Sampling-LP) to the value of (Sampling-OP). We showed above that for any feasible solution C to (Sampling-OP) we obtain a feasible solution (B, M) to (Sampling-LP) via (9) and $M_{i,j} = \min\{B_i, B_j\}$.

Fix $\delta > 0$. Then there exists $b_0 \in \mathbb{N}$ such that for all odd integers $b \geq b_0$

$$\sum_{i=0}^{N} e^{-(i+\frac{1}{2})\beta} \cdot B_i \;\leq\; (1+\delta) \cdot \sum_{i=0}^{N} \left(1-\tfrac{\beta}{b}\right)^{(i+\frac{1}{2})b} \cdot B_i \;\overset{(12)}{\leq}\; (1+\delta) \cdot \sum_{i=0}^{\infty} \left(1-\tfrac{\beta}{b}\right)^{i-1} \cdot C_i.$$

Thus the value of (Sampling-LP) is at most $(1+\delta)$ times the value of (Sampling-OP) with $p = \frac{\beta}{b}$ for all odd integers $b \geq b_0$. Hence, by Lemma 11, $(1+\delta)$ times the reciprocal of the optimum value of (Sampling-LP) is an upper bound on (2) for all $\frac{\beta}{b}$-normalized instances for all odd integers $b \geq b_0$. By Lemma 9, the same bound then holds for all instances with depot. Since this bound holds for all $\delta > 0$, it also holds for $\delta = 0$. ◀

2.3 The dual LP

In order to obtain a lower bound on the optimum value of (Sampling-LP), we provide a *feasible solution* to the *dual* linear program. For the dual LP, we introduce variables $x_{i,j}$ for the inequalities of type (14), variables $v_{i,j}$ and $w_{i,j}$ for the inequalities of type (15) and (16), respectively, and a variable y for inequality (13). Using these variables, the dual LP looks as follows:

$$\max \ \sigma^{-2} \cdot y \qquad \text{(Dual-Sampling-LP)}$$

$$\text{subject to} \quad 4\beta \cdot e^{-(i+j-1)\cdot\sigma\beta} \cdot y \ \leq \ v_{i,j} + w_{i,j} \quad \text{for } 0 \leq i \leq j \leq N \tag{18}$$

$$(\alpha+\delta_1) \cdot e^{-(k-1)\cdot\sigma\beta} \cdot y + \sum_{j=k}^{N} v_{k,j} + \sum_{j=0}^{k} w_{j,k} + \mathbb{1}_{k=1} \cdot \delta_2 \cdot y$$

$$+\mathbb{1}_{k>0} \cdot \left(\sum_{i=1}^{\min\{k,N-k\}} x_{i,k} + \sum_{i=k}^{N-k} x_{k,i} - \sum_{\substack{1 \leq i \leq j \leq N,\\ i+j=k}} x_{i,j} \right) \leq e^{-(k+\frac{1}{2})\beta} \quad \text{for } 0 \leq k \leq N \tag{19}$$

$$x, y, v, w \ \geq \ 0. \tag{20}$$

▶ **Corollary 16.** *Let N be an integer and $\beta > 0$. For any feasible solution (x, y, v, w) to* (Dual-Sampling-LP)*, σ^2/y is an upper bound on the approximation ratio of the sampling algorithm with $f(p) = 1-(1-p)^\sigma$ restricted to* A PRIORI *TSP instances with depot.* ⌟

We have computed a dual solution using Gurobi 10.0.1 with $\alpha = 1.5$, $\beta = \frac{1}{100}$, $N = 2500$, and $\sigma = 0.663$, yielding an upper bound of 3.094 and thus proving Theorem 3. The dual solution and a Python script that verifies that this is a feasible solution to (Dual-Sampling-LP) can be found at `https://doi.org/10.60507/FK2/JCUIRI`. For $\alpha = 1$, we get an upper bound of 2.694.

▶ **Comment 17.** Solving (Sampling-LP) with the same values for α, β, N, and σ yields an A PRIORI TSP instance of the same shape as the lower bound example provided in [4]. Hence we conjecture that the upper bound given by (Dual-Sampling-LP) converges to the lower bound given in Theorem 2 for $\beta \to 0$ and $N\beta \to \infty$.

2.4 Bounding the error term (Proof of Lemma 14)

We first prove the following auxiliary lemma:

▶ **Lemma 18.** *Let $n \in \mathbb{N}$ and $q \in (0,1)$. Then*

$$\sum_{k=n+1}^{\infty} k \cdot q^{k-1} \ = \ \frac{q^n}{(1-q)^2} \cdot (1+n-qn). \tag{21}$$

Proof. By induction on n. For $n = 0$, the statement is equivalent to the well-known formula

$$\sum_{k=1}^{\infty} k \cdot (1-q) \cdot q^{k-1} \ = \ \frac{1}{1-q}$$

for the expected value of a geometrically distributed random variable. Next, assume that (21) holds for some $n \in \mathbb{N}$. Then

$$\sum_{k=n+2}^{\infty} k \cdot q^{k-1} = \sum_{k=n+1}^{\infty} k \cdot q^{k-1} - (n+1) \cdot q^n \overset{(21)}{=} \frac{q^n}{(1-q)^2} \cdot (1+n-qn) - (n+1) \cdot q^n$$
$$= \frac{q^n}{(1-q)^2} \cdot (1+n-qn-(n+1) \cdot (1-q)^2) = \frac{q^{n+1}}{(1-q)^2} \cdot (n+2-q(n+1)),$$

which is (21) for $n+1$. ◀

Now we are ready to prove Lemma 14:

Proof of Lemma 14. We compute

$$2\beta \cdot \sum_{i,j \in \mathbb{Z}_{\geq 0} : \max\{i,j\} > N} e^{-(i+j-1)\cdot\sigma\beta} \cdot \min\{B_i, B_j\}$$
$$\leq 4\beta \cdot \sum_{i=0}^{N} \sum_{j=N+1}^{\infty} e^{-(i+j-1)\cdot\sigma\beta} \cdot B_i + 2\beta \cdot \sum_{i=N+1}^{\infty} \sum_{j=N+1}^{\infty} e^{-(i+j-1)\cdot\sigma\beta} \cdot B_i$$
$$= 4\beta \cdot \sum_{i=0}^{N} e^{-(i-1)\cdot\sigma\beta} \cdot B_i \cdot \sum_{j=N+1}^{\infty} e^{-j\sigma\beta} + 2\beta \cdot \sum_{i=N+1}^{\infty} e^{-(i-1)\cdot\sigma\beta} \cdot B_i \cdot \sum_{j=N+1}^{\infty} e^{-j\sigma\beta}$$
$$= \delta_1 \cdot \sum_{i=0}^{N} e^{-(i-1)\cdot\sigma\beta} \cdot B_i + \frac{2\beta}{e^{N\sigma\beta}(e^{\sigma\beta}-1)} \cdot \sum_{i=N+1}^{\infty} e^{-(i-1)\cdot\sigma\beta} \cdot B_i$$

Bounding $B_i \leq i \cdot B_1$ for $i > N$ by using to the triangle inequality (Proposition 12), we obtain

$$\alpha \cdot \sum_{k=N+1}^{\infty} e^{-(k-1)\cdot\sigma\beta} \cdot B_k + 2\beta \cdot \sum_{i,j \in \mathbb{Z}_{\geq 0} : \max\{i,j\} > N} e^{-(i+j-1)\cdot\sigma\beta} \cdot \min\{B_i, B_j\}$$
$$\leq \delta_1 \cdot \sum_{i=0}^{N} e^{-(i-1)\cdot\sigma\beta} \cdot B_i + \left(\alpha + \frac{2\beta}{e^{N\sigma\beta}(e^{\sigma\beta}-1)}\right) \cdot \sum_{k=N+1}^{\infty} e^{-(k-1)\cdot\sigma\beta} \cdot B_k$$
$$\leq \delta_1 \cdot \sum_{i=0}^{N} e^{-(i-1)\cdot\sigma\beta} \cdot B_i + \left(\alpha + \frac{2\beta}{e^{N\sigma\beta}(e^{\sigma\beta}-1)}\right) \cdot \sum_{k=N+1}^{\infty} e^{-(k-1)\cdot\sigma\beta} \cdot k \cdot B_1$$
$$= \delta_1 \cdot \sum_{i=0}^{N} e^{-(i-1)\cdot\sigma\beta} \cdot B_i + \delta_2 \cdot B_1,$$

where we used Lemma 18 in the final equality with $n = N$ and $q = e^{-\sigma\beta}$. ◀

3 Discussion

We conjecture (but could not prove) that our lower bound examples (cf. [4]) are really worst-case examples, and that the values of our linear programs converge to these bounds.

Another question is whether the master route ratio is $\frac{1}{1-e^{-1/2}}$ even for low-activity instances. Currently we only know the upper bound of 3 from [25], but know no example with master route ratio larger than $\frac{1}{1-e^{-1/2}}$ (and this value is attained by our example only as the activity tends to infinity). The analogous question applies to the sampling algorithm: whether we need to consider the low-activity case separately is an open question.

Finally, we hope that our approach can also help for proving a better bound for related problems where similar random sampling techniques are used, or for showing that known bounds are best possible.

References

1 Mohamed Abdellahi Amar, Walid Khaznaji, and Monia Bellalouna. An exact resolution for the probabilistic traveling salesman problem under the a priori strategy. *Procedia Computer Science*, 108:1414–1423, 2017. doi:10.1016/j.procs.2017.05.068.

2 Dimitris Bertsimas. *Probabilistic combinatorial optimization problems*. PhD thesis, Massachusetts Institute of Technology, 1988. URL: https://dspace.mit.edu/handle/1721.1/14386.

3 Dimitris J. Bertsimas, Patrick Jaillet, and Amedeo R. Odoni. A priori optimization. *Operations Research*, 38(6):1019–1033, 1990. doi:10.1287/opre.38.6.1019.

4 Jannis Blauth, Meike Neuwohner, Luise Puhlmann, and Jens Vygen. Improved guarantees for the a priori TSP, 2023. doi:10.48550/arXiv.2309.10663.

5 Neill E. Bowler, Thomas M. A. Fink, and Robin C. Ball. Characterization of the probabilistic traveling salesman problem. *Phys. Rev. E*, 68:036703, 2003. doi:10.1103/PhysRevE.68.036703.

6 Martijn van Ee, Leo van Iersel, Teun Janssen, and René Sitters. A priori TSP in the scenario model. *Discrete Applied Mathematics*, 250:331–341, 2018. doi:10.1016/j.dam.2018.04.002.

7 Martijn van Ee and René Sitters. The a priori traveling repairman problem. *Algorithmica*, 80(10):2818–2833, 2018. doi:10.1007/s00453-017-0351-z.

8 Friedrich Eisenbrand, Fabrizio Grandoni, Thomas Rothvoß, and Guido Schäfer. Connected facility location via random facility sampling and core detouring. *Journal of Computer and System Sciences*, 76(8):709–726, 2010. doi:10.1016/j.jcss.2010.02.001.

9 Finn Fernstrøm and Teresa Anna Steiner. A constant approximation algorithm for the uniform a priori capacitated vehicle routing problem with unit demands. *Information Processing Letters*, 159-160:105960, 2020. doi:10.1016/j.ipl.2020.105960.

10 Arun Ganesh, Bruce M. Maggs, and Debmalya Panigrahi. Robust algorithms for TSP and Steiner Tree. *ACM Trans. Algorithms*, 19(2), 2023. doi:10.1145/3570957.

11 Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 942–951. SIAM, 2008. URL: https://dl.acm.org/doi/10.5555/1347082.1347185.

12 Michel Goemans and Jon Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82(1):111–124, 1998. doi:10.1007/BF01585867.

13 Igor Gorodezky, Robert D. Kleinberg, David B. Shmoys, and Gwen Spencer. Improved lower bounds for the universal and a priori TSP. In Maria Serna, Ronen Shaltiel, Klaus Jansen, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 178–191. Springer, 2010. doi:10.1007/978-3-642-15369-3_14.

14 Anupam Gupta, Amit Kumar, Martin Pál, and Tim Roughgarden. Approximation via cost sharing: Simpler and better approximation algorithms for network design. *J. ACM*, 54(3), 2007. doi:10.1145/1236457.1236458.

15 Anupam Gupta, Amit Kumar, and Tim Roughgarden. Simpler and better approximation algorithms for network design. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 365–372. ACM, 2003. doi:10.1145/780542.780597.

16 Anupam Gupta, Martin Pál, R. Ravi, and Amitabh Sinha. Boosted sampling: Approximation algorithms for stochastic optimization. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 417–426. ACM, 2004. doi:10.1145/1007352.1007419.

17 Patrick Jaillet. Probabilistic traveling salesman problems. PhD thesis, Massachusetts Institute of Technology, 1985. URL: https://dspace.mit.edu/handle/1721.1/15231.

18 Patrick Jaillet. A priori solution of a traveling salesman problem in which a random subset of the customers are visited. *Operations Research*, 36(6):929–936, 1988. doi:10.1287/opre.36.6.929.

19 Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *J. ACM*, 50(6):795–824, 2003. doi:10.1145/950620.950621.

20 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing*, pages 32–45. ACM, 2021. doi:10.1145/3406325.3451009.

21 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A deterministic better-than-3/2 approximation algorithm for metric TSP. In Alberto Del Pia and Volker Kaibel, editors, *Integer Programming and Combinatorial Optimization*, pages 261–274. Springer, 2023. doi:10.1007/978-3-031-32726-1_19.

22 Fatemeh Navidi, Inge Li Gørtz, and Viswanath Nagarajan. Approximation algorithms for the a priori traveling repairman. *Operations Research Letters*, 48(5):599–606, 2020. doi:10.1016/j.orl.2020.07.009.

23 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993. doi:10.1287/moor.18.1.1.

24 Frans Schalekamp and David B. Shmoys. Algorithms for the universal and a priori TSP. *Operations Research Letters*, 36(1):1–3, 2008. doi:10.1016/j.orl.2007.04.009.

25 David Shmoys and Kunal Talwar. A constant approximation algorithm for the a priori traveling salesman problem. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Integer Programming and Combinatorial Optimization*, pages 331–343. Springer, 2008. doi:10.1007/978-3-540-68891-4_23.

26 Alejandro Toriello, William B. Haskell, and Michael Poremba. A dynamic traveling salesman problem with stochastic arc costs. *Operations Research*, 62(5):1107–1125, 2014. doi:10.1287/opre.2014.1301.

27 Laurence A. Wolsey. Heuristic analysis, linear programming and branch and bound. *Mathematical Programming Study*, 13:121–134, 1980. doi:10.1007/BFb0120913.

28 Anke van Zuylen. Deterministic sampling algorithms for network design. *Algorithmica*, 60(1):110–151, 2011. doi:10.1007/s00453-009-9344-x.

An FPT Algorithm for Splitting a Necklace Among Two Thieves

Michaela Borzechowski
Department of Mathematics and Computer Science, Freie Universität Berlin, Germany

Patrick Schnider
Department of Computer Science, ETH Zürich, Switzerland

Simon Weber
Department of Computer Science, ETH Zürich, Switzerland

Abstract

It is well-known that the 2-Thief-Necklace-Splitting problem reduces to the discrete Ham Sandwich problem. In fact, this reduction was crucial in the proof of the PPA-completeness of the Ham Sandwich problem [Filos-Ratsikas and Goldberg, STOC'19]. Recently, a variant of the Ham Sandwich problem called α-Ham Sandwich has been studied, in which the point sets are guaranteed to be well-separated [Steiger and Zhao, DCG'10]. The complexity of this search problem remains unknown, but it is known to lie in the complexity class UEOPL [Chiu, Choudhary and Mulzer, ICALP'20]. We define the analogue of this well-separation condition in the necklace splitting problem – a necklace is *n-separable*, if every subset A of the n types of jewels can be separated from the types $[n] \setminus A$ by at most n separator points. Since this version of necklace splitting reduces to α-Ham Sandwich in a solution-preserving way it follows that instances of this version always have unique solutions.

We furthermore provide two FPT algorithms: The first FPT algorithm solves 2-Thief-Necklace-Splitting on $(n-1+\ell)$-separable necklaces with n types of jewels and m total jewels in time $2^{O(\ell \log \ell)} + O(m^2)$. In particular, this shows that 2-Thief-Necklace-Splitting is polynomial-time solvable on n-separable necklaces. Thus, attempts to show hardness of α-Ham Sandwich through reduction from the 2-Thief-Necklace-Splitting problem cannot work. The second FPT algorithm tests $(n-1+\ell)$-separability of a given necklace with n types of jewels in time $2^{O(\ell^2)} \cdot n^4$. In particular, n-separability can thus be tested in polynomial time, even though testing well-separation of point sets is co-NP-complete [Bergold et al., SWAT'22].

2012 ACM Subject Classification Mathematics of computing → Combinatoric problems; Theory of computation → Computational geometry; Theory of computation → Fixed parameter tractability

Keywords and phrases Necklace splitting, n-separability, well-separation, ham sandwich, FPT

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.15

Related Version *Full Version*: https://arXiv.org/abs/2306.14508

Funding *Michaela Borzechowski*: DFG within the Research Training Group GRK 2434 *Facets of Complexity*.
Simon Weber: Swiss National Science Foundation under project no. 204320.

1 Introduction

The *necklace splitting problem* is one of the most famous problems in fair division. It is usually illustrated by the following story: two thieves have stolen a valuable necklaces with n different types of jewels (diamonds, rubies, etc.). They want to divide their bounty fairly between them, that is, in such a way that both of them get the same number of jewels of each type. As cutting through the necklace takes a lot of effort, they want to do this with as few cuts as possible. A mathematically inclined thief who knows the necklace splitting theorem [1, 3, 12] will realize that no matter how the jewels are ordered on the necklace, n cuts will always suffice for this. However, all known proofs of this result are of a topological nature and do

not give our thief any information on how to find the cuts. Thus, a more algorithmically inclined thief might wonder whether a set of n cuts can be found efficiently. Unfortunately, it turns out that the search problem of finding n cuts is in general PPA-complete [11], making an efficient algorithm unlikely. In this paper, we study separability conditions under which the thieves can find the cuts efficiently.

The ideas for the separability conditions stem from a variant of another famous fair division problem, namely the *Ham Sandwich problem*. The Ham Sandwich theorem [19] states that any d point sets (or mass distributions) in $\mathbb{R}^d$ can be simultaneously bisected by a single hyperplane. Again, finding such a Ham Sandwich Cut is in general PPA-complete [11]. However, under the assumption that the point sets are *well-separated* (which we will formally define in Section 2), the cut is unique [18] and the corresponding search problem lies in the complexity class UEOPL [5]. UEOPL is a subclass of PPA. It is conjectured to be a strict subclass, with a recent paper showing a black-box separation between the two classes [14].

The Ham Sandwich problem and the necklace splitting problem are intimately related. In fact, the necklace splitting theorem can be proved by lifting the necklace with n types of jewels to the *moment curve* in $\mathbb{R}^n$, which is the curve parameterized by $(t, t^2, t^3, \ldots, t^n)$, and then applying the Ham Sandwich theorem. By the same idea, the PPA-hardness for the Ham Sandwich problem follows from the PPA-hardness of the necklace splitting problem. In the well-separated setting, no hardness result is known for finding the now unique Ham Sandwich cut. A natural approach to show for example UEOPL-hardness of this problem would be to show hardness for a necklace splitting variant whose lifts give well-separated point sets. This leads to the definition of *n-separable necklaces*, which we again define formally in Section 2.

However, as we show in this paper, this approach will not work, as the necklace splitting problem on n-separable necklaces can be solved in polynomial time. Relaxing the notion of separability further, we get an FPT algorithm for the necklace splitting problem, parameterized by the separability:

▶ **Theorem 1.** *2-Thief-Necklace-Splitting can be solved in time $2^{O(\ell \log \ell)} + O(m^2)$ on every $(n-1+\ell)$-separable necklace C with n types of jewels and m total jewels.*

We also provide an FPT algorithm to check whether a necklace is $(n-1+\ell)$-separable. This is again in contrast to the Ham Sandwich problem, where it has been shown that checking well-separation is co-NP-complete [4].

Our work provides the first FPT viewpoint on the necklace splitting problem, which so far has only been studied from the viewpoint of approximation algorithms [2].

2 Preliminaries

2.1 Separability and Unique Solutions

▶ **Definition 2** (Necklace). *A* necklace *is a family C of disjoint finite point sets in $\mathbb{R}$. The sets in C are called* colors.

Note that in the literature, the points in each color $c \in C$ are also called *beads* or *jewels* of color c. Furthermore, this kind of necklace is sometimes also called an *open* necklace, since the colors are arranged in $\mathbb{R}$ and not on a cycle.

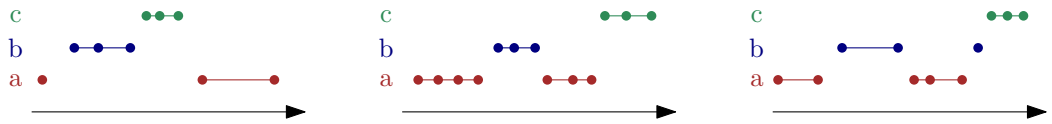
For simplicity, in the rest of this paper we assume that each color has an *odd* number of points. All of our results can be adapted to the more general setting without this restriction, or even to the setting where colors are finite unions of intervals. However, the definitions and proofs have to be adjusted carefully. We discuss these possible extensions of our results in the full version of the paper.

Since colors in a necklace are disjoint, we can view our necklace as a string over the alphabet C: each color defines one character and the sequence of characters is defined by the order in which the colors appear when going from $-\infty$ to ∞, with consecutive occurrences of the same color yielding just one character. See Figure 1 for an example.

We call the number of occurrences of a color c in this string the number of *components* it consists of. We say a color $c \in C$ is an *interval*, if it consists of exactly one component. In other words, a color c is an interval if its convex hull does not intersect any other color $c' \in C$. In Figure 1a, the green color c is an interval, whereas the red color a is not, it consists of two components.

▶ **Definition 3** (Separability). *A necklace C is* k-separable *if for all $A \subseteq C$ there exist k* separator points $s_1 < \ldots < s_k \in \mathbb{R}$ *that separate A from $C \setminus A$. More formally, if we alternatingly label the intervals $(-\infty, s_1], [s_1, s_2], \ldots, [s_{k-1}, s_k], [s_k, \infty)$ with A and $\overline{A}$, for every interval I labelled A we have $I \cap \bigcup_{c \in (C \setminus A)} c = \emptyset$ and for every interval I' labelled $\overline{A}$ we have $I' \cap \bigcup_{c \in A} c = \emptyset$.*

The separability *$sep(C)$ of a necklace C is the minimum integer $k \geq 0$ such that C is k-separable.*



(a) "$a\ b\ c\ a$" is 2-separable. **(b)** "$a\ b\ a\ c$" is 3-separable. **(c)** "$a\ b\ a\ b\ c$" is 4-separable.

Figure 1 Necklace C with 3 colors a, b and c.

Note that for a necklace with n colors, $sep(C) \geq n - 1$, and this is tight, as can be seen in Figure 1a. Our definition of k-separability is strongly related to the well known notion of *well-separation*.

▶ **Definition 4.** *Let $P_1, \ldots, P_k \subset \mathbb{R}^d$ be point sets. They are* well-separated *if and only if for every non-empty index set $I \subset [k]$, the convex hulls of the two disjoint subfamilies $\bigcup_{i \in I} P_i$ and $\bigcup_{i \in [k] \setminus I} P_i$ can be separated by a hyperplane.*

A set of two colors in $\mathbb{R}$ is 1-separable if and only if it is well-separated. Furthermore we observe the following property.

▶ **Lemma 5.** *Let C be a set of n colors in $\mathbb{R}$. Let C' be the set of subsets of $\mathbb{R}^n$ obtained by lifting each point in each color of C to the n-dimensional moment curve using the function $f(t) = (t, t^2, \ldots, t^n)$. Then the set C is n-separable if and only if C' is well-separable.*

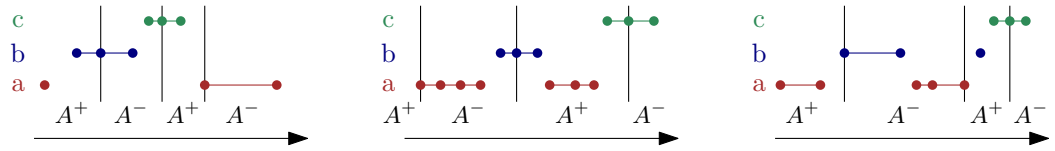
Proof. If C is n-separable, for each subset A of C, there exist n points $S = (s_1, \ldots, s_n)$ partitioning C into intervals alternatingly labelled A and $\overline{A}$. Let H be the hyperplane that goes through these separator points S lifted to the moment curve. By [15, Lemma 5.4.2], at each separating point, the moment curve passes from one side of H to the other. The points belonging to intervals labelled A lie on one side of the hyperplane and the points belonging to intervals labelled $\overline{A}$ lie on the other side. Since this holds for all subsets of C, it follows that C' is well-separated.

If C' is well-separated, for each subset A' of colors, there exists a hyperplane that separates A' from $C' \setminus A'$. By [15, Lemma 5.4.2], this hyperplane intersects the moment curve at at most n points. These points define the separator points that show that C is n-separable. ◀

The problem of *Necklace Splitting* is that two thieves want to split the necklace they stole into equal parts with as few cuts as possible. Mathematically we partition the necklace into several intervals which belong to each thief in turn.

▶ **Definition 6** (2-Thief-Necklace-Splitting). *Given a necklace C with n colors, find n* split points *that split the necklace into $n+1$ open intervals alternatingly labelled A^+ and A^-, such that for each color $c \in C$, the union of all intervals labelled A^+ contains the same number of points of c as the union of all intervals labelled A^-.*

It is well known that there always exists a solution to this problem [1, 3, 12]. Note that due to our assumption of every color containing an odd number of points, every solution must contain exactly one point per color as a split point.



(a) Solution for "*a b c a*". **(b)** Solution for "*a b a c*". **(c)** Solution for "*a b a b c*".

Figure 2 Example of solutions to 2-Thief-Necklace-Splitting.

▶ **Theorem 7.** *Let C be an n-separable necklace with n colors. There is a unique solution to 2-Thief-Necklace-Splitting on C.*

In order to prove the above theorem, we consider the classical reduction of 2-Thief-Necklace-Splitting to the Ham-Sandwich problem obtained by lifting the points to the moment curve, as it appeared in many works before [7, 11, 15, 17]. However, since the necklace we apply this reduction to is n-separable, by Lemma 5, the resulting points are well-separated, which allows us to apply the following stronger version of the Ham-Sandwich theorem due to Steiger and Zhao [18].

▶ **Lemma 8** (α-Ham-Sandwich Theorem, [18]). *Let $P_1, \ldots, P_n \subset \mathbb{R}^n$ be finite well-separated point sets in* weak general position[1]*, and let $\alpha_1, \ldots, \alpha_n$ be positive integers with $\alpha_i \leq |P_i|$, then there exists a unique $(\alpha_1, \ldots, \alpha_n)$-cut, i.e., a hyperplane H that contains a point from each color and such that for the closed positive halfspace H^+ bounded by H we have $|H^+ \cap P_i| = \alpha_i$.*

Proof of Theorem 7. We lift all the points in C to the moment curve. The points are in general position [15] (and thus also in weak general position). By Lemma 5 if C is n-separable, then the point sets lifted to the moment curve are well-separated.

By the α-Ham-Sandwich theorem there exists a unique $(\lceil \frac{|c_1|}{2} \rceil, \ldots, \lceil \frac{|c_n|}{2} \rceil)$-cut that halves all colors. This cut is a hyperplane H that goes through n lifted points, one point of each color. These points define a solution $Q = (q_1, \ldots, q_n)$ of 2-Thief-Necklace-Splitting.

Assume that the solution Q is not unique, i.e., there is another solution $Q' \neq Q$ to C. The points Q' lifted to the moment curve define another hyperplane $H' \neq H$ with one point of each color, which is also a $(\lceil \frac{|c_1|}{2} \rceil, \ldots, \lceil \frac{|c_n|}{2} \rceil)$-cut. But by Lemma 8 there is a unique hyperplane with this property, so Q' cannot exist. ◀

[1] Weak general position is a condition that requires only subsets of the points of the form $\{p_1, \ldots, p_n, p_{n+1}\}$ for $p_i \in P_i$ for $1 \leq i \leq n$ and $p_{n+1} \in P_1 \cup \ldots \cup P_n$ to be in general position.

In this proof, we do not use the property that Lemma 8 guarantees that there is a solution for *every* choice of α, we merely use it for the guaranteed uniqueness of a solution for a halving cut.

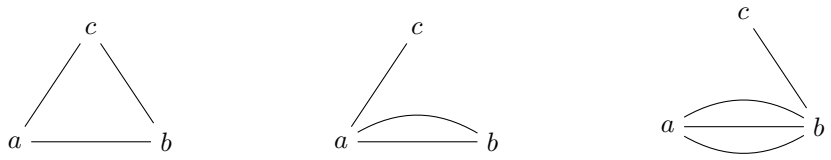
Note that the opposite direction of Theorem 7 does not hold, i.e., there are necklaces with n colors which are not n-separable but still have unique solutions for 2-Thief-Necklace-Splitting, see Figure 2c for an example.

2.2 Graph-Theoretic Aspects

To argue about the separability of necklaces, we wish to think about graphs rather than strings or even point sets. For every necklace, we thus define its walk graph:

▶ **Definition 9** (Walk graph). *Given a necklace C, the walk graph G_C is the multigraph with $V = C$ and with every potential edge $\{a, b\} \in \binom{V}{2}$ having multiplicity equal to the number of substrings "ab" plus the number of substrings "ba" in the string describing C.*

The walk graphs of the example necklaces in Figure 1 can be seen in Figure 3.



(a) Walk graph for "$a\ b\ c\ a$". **(b)** Walk graph for "$a\ b\ a\ c$". **(c)** Walk graph for "$a\ b\ a\ b\ c$".

Figure 3 Walk graphs of the examples in Figure 1.

Note that given a necklace C as a set of point sets, both the string describing it as well as the walk graph can be built in linear time in the size of the necklace $\sum_{c \in C} |c|$.

Recall that a graph is *Eulerian* if it contains a Eulerian tour, a closed walk that uses all edges exactly once. A graph is *semi-Eulerian* if it contains a Eulerian path, a (not necessarily closed) walk that uses all edges exactly once.

▶ **Observation 10.** *The walk graph of a necklace is connected and semi-Eulerian.*

Recall the following well-known fact about semi-Eulerian graphs.

▶ **Lemma 11.** *In a semi-Eulerian (multi-)graph, at most two vertices have odd degrees.*

The separability of a necklace turns out to be equivalent to the max-cut in its walk graph.

▶ **Definition 12** (Cut). *In a (multi-)graph G on the vertices V, a* cut *is a subset $A \subseteq V$. The* size $\mu(A)$ of a cut *A is the number of edges $\{u, v\}$ in G such that $u \in A$ and $v \notin A$. The* max-cut*, denoted by $\mu(G)$, is the largest size of any cut $A \subseteq V$.*

▶ **Lemma 13.** *For every necklace C, we have $sep(C) = \mu(G_C)$.*

Proof. For every subset $A \subseteq C$, the number of separator points needed to separate the colors in A from $C \setminus A$ is given by the size of the cut A in G_C, since the edges going over this cut correspond one to one to the points in the necklace where the necklace switches from a color in A to a color not in A, or vice versa. Thus, the max-cut $\mu(G_C)$ corresponds to the maximal number of separator points we need to separate any two subsets of colors. ◀

In our proofs we will often show that certain structures or properties do not appear in walk graphs of necklaces with bounded separability. The general strategy for these proofs will be to show that walk graphs with these structures or properties have a large max-cut, and thus the corresponding necklaces cannot have the claimed separability. Our main tool for this is the following bound, originally conjectured by Erdős [10] and proven by Edwards [8, 9].

▶ **Theorem 14** (Edwards-Erdős bound). *A simple connected graph G with n vertices and m edges has a maximum cut $\mu(G)$ of at least $\omega(G) := \frac{m}{2} + \frac{n-1}{4}$.*

Since walk graphs are not simple graphs, we will use a corollary of the following strengthening, due to Poljak and Turzík [16]:

▶ **Theorem 15** ([16]). *For a connected graph G with weight function $w : E \to \mathbb{R}_+$, there exists a cut of weight at least*

$$\frac{\sum_{e \in E} w(e)}{2} + \frac{t(G,w)}{4},$$

where $t(G,w)$ is the weight of a minimum-weight spanning tree of G.

▶ **Corollary 16.** *A connected (multi-)graph G with n vertices and m edges has a maximum cut $\mu(G)$ of at least $\omega(G) := \frac{m}{2} + \frac{n-1}{4}$.*

For determining the separability of a necklace, we will use an algorithm due to Crowston, Jones and Mnich [6] to decide max-cut beyond the Edwards-Erdős bound.

▶ **Theorem 17** (FPT algorithm [6]). *There exists an algorithm that decides whether for a given simple connected graph G with n vertices and m edges the max-cut $\mu(G)$ is at most $\omega(G) + k$ in time $2^{O(k)} \cdot n^4$.*

This is a so-called fixed-parameter algorithm; for any *fixed* parameter k, the algorithm runs in polynomial time in n, with the exponent not depending on k. Note again that this algorithm only works on simple graphs, thus, we will need to alter the walk graphs to be able to apply this algorithm.

3 An FPT Algorithm for 2-Thief-Necklace-Splitting

In this section we show Theorem 1:

▶ **Theorem 1.** *2-Thief-Necklace-Splitting can be solved in time $2^{O(\ell \log \ell)} + O(m^2)$ on every $(n-1+\ell)$-separable necklace C with n types of jewels and m total jewels.*

The algorithm we use is recursive, based on the following crucial observation.

▶ **Theorem 18.** *Let C be an $(n-1+\ell)$-separable necklace with n colors. If $n \geq 6\ell + 2$ there must exist*

- **(i)** *two neighboring colors that are both intervals, or*
- **(ii)** *one color that only consists of exactly two components.*

Proof. Since the walk graph is semi-Eulerian, it contains either 0 or 2 vertices with odd degree (recall Observation 10 and Lemma 11). A color that is an interval has degree 2, unless it is at the beginning or end of the necklace. A color that consists of more than two components has degree at least 6 (or 5 or 4 if it is at the beginning and/or end of the necklace).

Let $A \subseteq C$ be the set of intervals. Note that if no two intervals are neighboring, we can pick all the intervals as a cut A, which has size at least $\mu(A) \geq 2|A| - 2$. Since we know that $\mu(G_C) \leq n - 1 + \ell$, we must have that $|A| \leq \frac{n+1+\ell}{2}$.

Assume that the theorem does not hold, and that there thus exist no neighboring intervals and no color consisting of exactly two components. We can then bound the sum of degrees $\sum_{c \in C} deg(c) \geq 2 \cdot \frac{n+1+\ell}{2} + 6 \cdot (n - \frac{n+1+\ell}{2}) - 2 = 4n - 2\ell - 4$. Thus, the number of edges $|E|$ in G_C is bounded $|E| \geq \frac{4n-2\ell-4}{2} = 2n - \ell - 2$. Due to Corollary 16 we thus get that $\mu(G_C) \geq \frac{2n-\ell-2}{2} + \frac{n-1}{4} = \frac{5}{4}n - \frac{\ell}{2} - \frac{5}{4}$. By the assumption $n \geq 6\ell + 2$, we therefore have $\mu(G_C) \geq n - \frac{3}{4} + \ell$, which is a contradiction to the assumption that $\mu(G_C) \leq n - 1 + \ell$. Thus, the theorem follows. ◀

To use Theorem 18 to recursively solve smaller instances, we need to make sure that the separability of the smaller instances translates back to the separability of the original instance. The following two lemmas provide this necessary correspondence.

▶ **Lemma 19.** *Let C be a necklace. Let C' be the necklace obtained by removing two neighboring intervals c, c' from C. Then, $sep(C') = sep(C) - 2$.*

Proof. In the walk graph, removing two neighboring intervals corresponds to replacing a path (a, c, c', b) of length 3 by a direct edge connecting a and b.

Every cut $A' \subseteq C'$ in $G_{C'}$ of size k can be extended to a cut $A \subseteq C$ in G_C of size $k + 2$: For every vertex $v \in C'$, we have $v \in A'$ iff $v \in A$. Furthermore, $c \in A$ iff $a \notin A'$ and $c' \in A$ iff $b \notin A'$. Thus, $sep(C) \geq sep(C') + 2$.

Similarly, every cut $A \subseteq C$ in G_C of size k induces a cut $A' = A \cap C'$ of size at least $k - 2$ in $G_{C'}$. Thus, $sep(C') \geq sep(C) - 2$, and we get $sep(C') = sep(C) - 2$. ◀

▶ **Lemma 20.** *Let C be a necklace on n colors that is $(n - 1 + \ell)$-separable. The necklace C' obtained by reducing a color $c \in C$ to a subset $\emptyset \subset c' \subset c$ is still $(n - 1 + \ell)$-separable.*

Proof. By simplifying a necklace, we cannot increase its separability. ◀

We are now ready to present Algorithm 1, an FPT algorithm to solve 2-Thief-Necklace-Splitting on $(n-1+\ell)$-separable necklaces. The strategy is to reduce the given necklace either by removing two neighboring intervals, or by removing one of the two components in a color that consists of exactly two components. By Lemmas 19 and 20, if C is $(n-1+\ell)$-separable, the resulting necklace C' is again $(n' - 1 + \ell)$-separable (for $n' = |C'|$), and can thus be solved recursively. The solution of the reduced case is then extended back to a solution of the original necklace. A necklace can be reduced as long as Theorem 18 applies, and thus we only need to solve the case $n < 6\ell + 2$ directly.

For an example of the execution of the algorithm, see Figure 4 and Figure 5. Note that these small instances would technically be solved by brute-force and merely serve as illustrations.

Proof of Theorem 1. We first argue for correctness of Algorithm 1. By Theorem 18, if we reach line 8 we can always find a color which consists out of exactly two components, so the algorithm can never fail to finish.

We have to argue that our algorithm returns a correct solution in both line 6 and line 15.

(i) Line 6: The constructed solution splits the two neighboring intervals correctly. Since we place two splits, the parity of the partition outside of these intervals does not change in comparison to the solution Q obtained recursively. Thus, all other colors are also split correctly.

Algorithm 1 RecursiveNS.

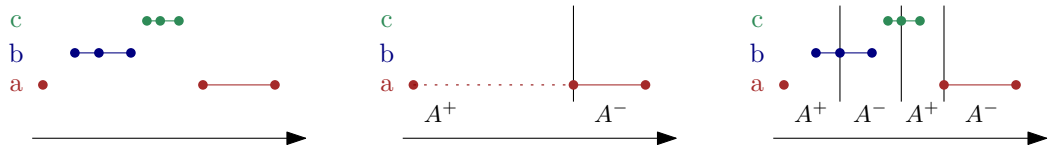
Input: An $(n-1+\ell)$-separable necklace C with n colors.
Output: n split points.

```
1:  if n < 6ℓ + 2 then
2:      Q ← BruteForce(C)
3:      return Q
4:  else if there exist two neighboring intervals c, c' ∈ C then
5:      Q ← RecursiveNS(C \ {c, c'})
6:      return Q ∪ {median(c), median(c')}
7:  else
8:      c ← a color consisting of two components c_1, c_2
9:      c' ← largest component of c
10:     if |c'| is even then
11:         Add a median point to c'
12:     Q ← RecursiveNS((C \ {c}) ∪ {c'})
13:     {q} ← Q ∩ c'
14:     q' ← q shifted right/left by ⌈min(|c_1|,|c_2|) / 2⌉ points of c' ▷ direction depending on parity
        of number of split points in Q between c_1 and c_2
15:     return Q \ {q} ∪ {q'}
```

(ii) Line 15: The constructed solution splits color c correctly, and q' lies in the same component of c as q, since c' is the larger of the two components. Shifting the split within the same component of c does not change the partition outside of this component in comparison to the solution Q obtained recursively. Thus, all other colors are also split correctly.

It remains to argue for the runtime of Algorithm 1. Clearly, we only use the brute-force approach at line 2 once. In an $(n-1+\ell)$-separable necklace with $n < 6\ell+2$, each color has at most $O(\ell)$ components. For each guess of one component per color, it can be determined in polynomial time in ℓ whether this guess admits a solution. There are at most $\ell^{O(\ell)}$ guesses, thus we can solve this base case in time $2^{O(\ell \log \ell)}$.

In the rest of the algorithm, on each level of the recursion we reduce the number of points in the necklace by at least one, and we can make the necessary adjustments and find the needed colors in linear time in the number of points. Thus, the total runtime of the algorithm is $2^{O(\ell \log \ell)} + O(\sum_{c \in C} |c|)^2$, as claimed. ◀

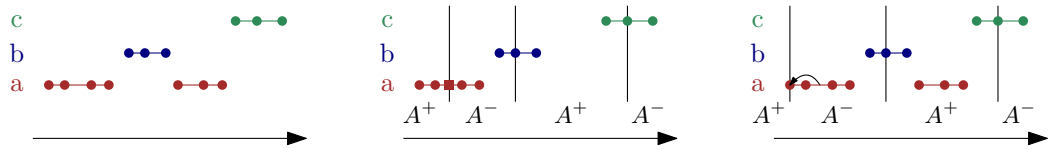


(a) Original necklace. **(b)** Reduced necklace. **(c)** Solution.

Figure 4 Example step of Algorithm 1 using the reduction of removing two neighboring intervals (b and c).

For the special case of n-separable necklaces, i.e., $\ell = 1$, we get the following corollary:

▶ **Corollary 21.** *Finding the unique solution for 2-Thief-Necklace-Splitting on an n-separable necklace with n colors takes polynomial time.*



(a) Original necklace. **(b)** Reduced necklace. **(c)** Solution.

Figure 5 Example step of Algorithm 1 using the reduction of removing a component from the two-component color a.

Until now, both Theorem 1 and Corollary 21 work under the initial promise that C is $(n-1+\ell)$-separable (or n-separable respectively). If the algorithm fails because none of the cases applies, this certifies that the input necklace was *not* $(n-1+\ell)$-separable. On the other hand, Algorithm 1 may run successfully, even if the input necklace is not $(n-1+\ell)$-separable, and if it does run successfully, its output is always a correct solution. Since Algorithm 1 can produce these "false positives", it cannot be used to decide $(n-1+\ell)$-separability. We tackle that problem in the next section.

4 Testing Separability

At first, it seems like finding a polynomial-time algorithm for deciding whether a necklace is $(n-1+\ell)$-separable may be futile, since we have the following theorem due to Guruswami [13]:

▶ **Theorem 22** ([13]). *Given a Eulerian graph G and an integer k, deciding whether the size of the max-cut $\mu(G) \geq k$ is* NP*-complete.*

Since to compute the separability of a necklace we need to compute the max-cut of its walk graph, and since every Eulerian graph is the walk graph of some necklace[2], we get the following corollary:

▶ **Corollary 23.** *Given a necklace C of n colors and an integer k, deciding whether C is k-separable is* co-NP*-complete.*

However, not all hope is lost. To check whether a necklace is $(n-1+\ell)$-separable, we do not need to compute the max-cut of its walk graph, we merely need to check whether it is at most $(n-1+\ell)$. We next provide an FPT algorithm that checks $(n-1+\ell)$-separability for fixed parameter ℓ. With $\ell = 1$ this shows that testing n-separability of n colors is solvable in polynomial time, even though both testing k-separability of n colors with k as input as well as testing well-separation of point sets are co-NP-complete [4]. More generally, we show the following theorem:

▶ **Theorem 24.** *There exists an FPT algorithm for fixed parameter ℓ that can decide whether the max-cut of a given semi-Eulerian multigraph G_C with n vertices is at most $n-1+\ell$, i.e., it can decide whether $\mu(G_C) \leq n-1+\ell$ in time $2^{O(\ell^2)} \cdot n^4$.*

By Theorem 17, there exists an algorithm that decides whether a *simple* graph G with n vertices and a fixed parameter k has a max-cut of size $\mu(G) \leq \omega(G) + k = \frac{|E(G)|}{2} + \frac{n-1}{4} + k$ in $2^{O(k)} \cdot n^4$ time. But our input graph G_C is a multigraph and we have no bound on its number of edges, nor on the distance between $\omega(G_C)$ and $n-1+\ell$. In order to use this algorithm to decide separability, we need the following:

[2] Simply find a Eulerian path through the graph and place one point per character in the respective color. If some color has an even number of points, add one more to an existing component.

1. Derive a graph G'_C from G_C such that we *can* lower bound $|E(G'_C)|$ and thus $\omega(G'_C)$.
2. Prove that there is a bounded number of multi-edges in G'_C.
3. Transform G'_C into a simple graph G''_C by blowing up its multi-edges by a constant factor.

In the following, we will use the term *interval* for intervals on necklaces as well as their corresponding vertices interchangeably. The intervals in C correspond to the vertices in G_C with degree at most 2, except for possibly one vertex of degree 2 that is both the starting and ending point of the fixed Eulerian path; this single vertex does not correspond to an interval.

▶ **Lemma 25.** *Given a semi-Eulerian multigraph G on n vertices, we can either detect that $\mu(G) > n-1+\ell$, or we can build a multigraph G' on n' vertices such that $|E(G')| \geq \frac{3}{2}n' - \frac{\ell}{2} - 1$, and such that $\mu(G) \leq n-1+\ell$ if and only if $\mu(G') \leq n'-1+\ell$.*

Proof. Given a multigraph G, let G' be the result of applying Lemma 19 on G exhaustively. As long as there are two adjacent intervals in G, we can remove the two intervals, thus reducing the maximum cut size by 2. In each such step we remove 2 vertices, 3 edges and add 1 new edge. Thanks to Lemma 19, we have the desired correspondence between $\mu(G)$ and $\mu(G')$.

Assume there are at least $\frac{n'+\ell}{2}+1$ intervals in G'. Then the cut A in G' with all intervals on one side and all other vertices on the other side has size $\mu(A) \geq 2\cdot(\frac{n'+\ell}{2}+1)-2 = n'+\ell$. It follows that $\mu(G') \geq \mu(A) > n'-1+\ell$. In this case we can thus detect that $\mu(G) > n-1+\ell$.

In the other case, there are strictly fewer than $\frac{n'+\ell}{2}+1$ intervals in G'. All other vertices have degree at least 4 (excluding the start and end vertex). Therefore the sum of degrees in G' is

$$\sum_{v \in V(G')} deg(v) \geq \frac{n'+\ell}{2}\cdot 2 + \frac{n'-\ell}{2}\cdot 4 - 2 = 3n' - \ell - 2.$$

Thus the number of edges in G' is $|E(G')| \geq \frac{3}{2}n' - \frac{\ell}{2} - 1$. ◀

We can now see the following.

▶ **Observation 26.** *Given this bound on $|E(G')|$, the bound $\omega(G')$ given by Corollary 16 can be bounded by*

$$\omega(G') \geq \frac{\frac{3}{2}n' - \frac{\ell}{2} - 1}{2} + \frac{n'-1}{4} = n' - \frac{\ell}{4} - \frac{3}{4}.$$

Thus, by the process of eliminating neighboring intervals, we have managed to get the difference between $(n'-1+\ell)$ and $\omega(G'_C)$ to be a constant depending only on ℓ.

Next we show that the total multiplicity M of the multi-edges in G'_C cannot be too large. We show that if G'_C has maximum cut size at most $n'-1+\ell$, the total multiplicity of multi-edges can be bounded by a function solely depending on ℓ, and not n or $|E(G'_C)|$.

▶ **Lemma 27.** *In a multigraph G on n vertices with $\mu(G) \leq n-1+\ell$, the total multiplicity of the multi-edges in G is at most $2\ell^2$.*

Proof. Let G' be a weighted simple graph with an edge of weight $m-1$ for every multi-edge of multiplicity $m \geq 2$ in the graph G. Note that the total weight of G' is at least half of the total multiplicity of multi-edges in G.

Let F be a spanning forest in G' with total weight w. Given F, we can build a spanning tree T of G of total weight $n-1+w$, since every edge of F of weight $m'-1$ corresponds to a multi-edge of multiplicity m' in G, and all additional edges used to make F into a spanning

tree have weight 1. Since every tree is bipartite, the weight of T is a lower bound on the max-cut of G: $\mu(G) \geq n - 1 + w$. Thus, for a given G with $\mu(G) \leq n - 1 + \ell$, the total weight of F must be at most ℓ.

We thus only need to show that in a simple weighted graph (in our case, G'), in which every weight is at least 1, and whose maximum-weight spanning forest has weight at most ℓ, the total weight of the graph is at most ℓ^2. To see this, we successively remove spanning forests from G' until G' is empty. Every spanning forest we remove has weight at most ℓ. As every edge has weight at least 1, every vertex in G' has degree at most ℓ. Thus, we are done after removing at most ℓ spanning forests. Thus, the total weight of G' is at most ℓ^2.

We conclude that the total multiplicity of multi-edges in G can be at most $2\ell^2$. ◀

Finally, we show how G'_C can be transformed into a simple graph G''_C. Let a and b be vertices in G'_C with a multi-edge of multiplicity m between them. We construct the graph G''_C from G'_C by removing the multi-edge between a and b and introducing m paths of length three from a to b, all going through separate vertices. See Figure 6 for an example application of this process.

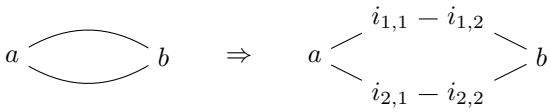


Figure 6 Example of blowing up a multi-edge of multiplicity 2 to make the graph simple.

This process is again constructed in such a way that the change of the max-cut is predictable:

▶ **Lemma 28.** *Let G be a multigraph on n vertices. Let a and b be vertices in G with a multi-edge of multiplicity m between them. Let G' be the result of blowing up the multi-edge between a and b in G. Then, $\mu(G') = \mu(G) + 2m$.*

Proof. Let $A \subseteq V(G)$ be some max-cut in G with $\mu(A) = \mu(G)$.

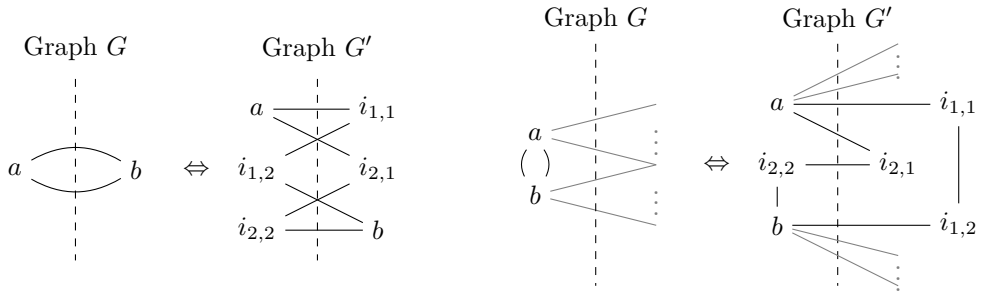
We distinguish between two cases. If the multi-edge goes across the cut, i.e. $a \in A$ and $b \notin A$, the same cut in G' has m fewer edges (namely the multi-edge) and $3m$ edges more, namely all of the newly introduced edges of the paths, see Figure 7a. If the multi-edge between a and b is *not* in the max-cut of G, there is a cut in G' that has $2m$ new edges, namely one of each newly introduced path, see Figure 7b. Thus, a max-cut of size $\mu(G)$ in G implies a cut of size $\mu(G) + 2m$ in G', and thus $\mu(G') \geq \mu(G) + 2m$.

For the other direction, consider a max-cut A' of G'. Since A' is maximal, it must either contain all $3m$ intermediate edges between a and b, and put a and b on different sides of the cut, or it must put a and b on the same side of the cut, and contain exactly $2m$ intermediate edges (see again Figure 7). Thus, there must exist a cut A in G which contains exactly $2m$ fewer edges than A', and we get $\mu(G) \geq \mu(G') - 2m$.

We conclude that $\mu(G') = \mu(G) + 2m$. ◀

We are now ready to put this all together and describe the algorithm proving Theorem 24.

Proof of Theorem 24. We prove that Algorithm 2 is correct and runs in time $2^{f(\ell)} \cdot n^4$. Correctness follows from Lemma 25, Lemma 27 and Lemma 28. Clearly, all steps except the invocation of the FPT algorithm of Theorem 17 in the last line can be performed in $O(n^2 + \ell^2)$.



(a) Case 1: The multi-edge is in max-cut of G. **(b)** Case 2: The multi-edge is *not* in max-cut of G.

Figure 7 Change of max-cut size when blowing up a multi-edge of multiplicity 2.

Algorithm 2 FPT algorithm for testing $\mu(G_C) \leq n - 1 + \ell$ with fixed parameter ℓ.

Input: A semi-Eulerian multigraph G_C on n vertices.
Output: True iff $\mu(G_C) \leq n - 1 + \ell$.

$G'_C \leftarrow G_C$
while there exist neighboring intervals in G'_C **do**
  Remove two neighboring intervals from G'_C.
$n' \leftarrow |V(G'_C)|$
$i \leftarrow$ The number of intervals in G'_C.
if $i > \frac{n'+\ell}{2}$ **then return** false ▷ *based on Lemma 25*
$M \leftarrow$ The total multiplicity of multi-edges in G'_C.
if $M > 2\ell^2$ **then return** false ▷ *based on Lemma 27*
$G''_C \leftarrow$ The result of applying Lemma 28 to every multi-edge in G'.
return $\mu(G''_C) \leq (n' - 1 + \ell) + 2M$ ▷ *using FPT algorithm of Theorem 17.*

We choose k such that when we call the FPT algorithm of Theorem 17 with G''_C and k it decides $\mu(G''_C) \leq (n' - 1 + \ell) + 2M$, i.e., we choose k such that $(n' - 1 + \ell) + 2M = \omega(G''_C) + k$. Therefore let $k := ((n' - 1 + \ell) + 2M) - \omega(G''_C)$.

For bounding the runtime of this invocation, we need to check that k is dependent only on ℓ. Recall that by Observation 26 we can bound $(n' - 1 + \ell) - \omega(G'_C) \leq \frac{5}{4}\ell - \frac{1}{4}$, a quantity depending only on our parameter ℓ. To relate $\omega(G''_C)$ to $\omega(G'_C)$, we can see that blowing up a multi-edge in G' of multiplicity m adds $2m$ edges and $2m$ vertices and thus changes ω by $\frac{2m}{2} + \frac{2m}{4} = \frac{3}{2}m$. Thus we have $\omega(G''_C) = \omega(G'_C) + \frac{3}{2}M$. We can now put everything together and get $k \leq 2M + \frac{5}{4}\ell - \frac{1}{4} - \frac{3}{2}M = \frac{1}{2}M + \frac{5}{4}\ell - \frac{1}{4}$, and since $M \leq \ell^2$, we get that k is bounded by $O(\ell^2)$. Thus, the final invocation of the algorithm of Theorem 17 runs in time $2^{O(\ell^2)} \cdot n^4$. ◀

5 Conclusion and Further Directions

In conclusion, we proved that 2-Thief-Necklace-Splitting on n-separable necklaces has a unique solution and can be solved in polynomial time. Also n-separability can be tested in polynomial time. Furthermore, we showed that 2-Thief-Necklace-Splitting, which in general is known to be PPA-complete, admits an FPT algorithm for the parameter ℓ such that the input necklace is $(n - 1 + \ell)$-separable. Lastly, we showed that testing $(n - 1 + \ell)$-separability is also FPT, even though testing well-separation of point sets in $\mathbb{R}^n$ is co-NP-complete.

The condition of n-separability is only sufficient for uniqueness of the solution to 2-Thief-Necklace-Splitting. An interesting followup question is whether there also exist necessary conditions for such uniqueness.

As our main open question we wonder how our algorithm for 2-Thief-Necklace-Splitting can be extended to more general settings. Firstly, can we also find polynomial time algorithms for k-Thief-Necklace-Splitting under the constraint of n-separability? Secondly, instead of halving every color class, can we maybe find an algorithm to find any $(\alpha_1, \dots, \alpha_n)$-cut? The existence of these cuts is also guaranteed by Lemma 8, however our algorithm really only works for halving, since if we are not halving, the solution is not guaranteed to split a color with two components in the bigger component.

Another interesting followup question is whether one can lift the definition of k-separability into higher dimensions. In other words, for n point sets $P = \{P_1, \dots, P_n\}$ in $\mathbb{R}^d$, can each subset A of P be separated from $P \setminus A$ by k hyperplanes? Well-separation then becomes 1-separability. Thus, deciding k-separability for k as input or even for the case $k = 1$ is co-NP-hard. It is likely that special cases such as d-separability or n-separability are also hard to decide. While well-separation is also contained in co-NP, this is not clear for k-separability for $k > 1$. Like 2-Thief-Necklace-Splitting, which has a unique solution under the condition of n-separability, one could also investigate whether there are other geometric problems which gain interesting properties under the condition of the input being well-separated, or k-separable for some k.

Finally, can we extend our FPT algorithm for deciding $\mu(G) \leq n - 1 + \ell$ on semi-Eulerian multigraphs to work on all connected multigraphs? Furthermore, can we maybe also decide $\mu(G) \leq \omega(G) + \ell$ (to get a direct analogue of the algorithm of Crowston, Jones, and Mnich for multigraphs) and not just $\mu(G) \leq n - 1 + \ell$?

References

1. Noga Alon. Splitting necklaces. *Advances in Mathematics*, 63(3):247–253, 1987. doi:10.1016/0001-8708(87)90055-7.
2. Noga Alon and Andrei Graur. Efficient splitting of necklaces. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.14.
3. Noga Alon and Douglas B. West. The Borsuk-Ulam theorem and bisection of necklaces. *Proceedings of the American Mathematical Society*, 98(4):623–628, 1986. doi:10.1090/S0002-9939-1986-0861764-9.
4. Helena Bergold, Daniel Bertschinger, Nicolas Grelier, Wolfgang Mulzer, and Patrick Schnider. Well-Separation and Hyperplane Transversals in High Dimensions. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*, volume 227 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2022.16.
5. Man-Kwun Chiu, Aruni Choudhary, and Wolfgang Mulzer. Computational Complexity of the α-Ham-Sandwich Problem. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.31.
6. Robert Crowston, Mark Jones, and Matthias Mnich. Max-cut parameterized above the Edwards-Erdős bound. *Algorithmica*, 72(3):734–757, 2015. doi:10.1007/s00453-014-9870-z.

7 Jesús De Loera, Xavier Goaoc, Frédéric Meunier, and Nabil Mustafa. The discrete yet ubiquitous theorems of Carathéodory, Helly, Sperner, Tucker, and Tverberg. *Bulletin of the American Mathematical Society*, 56(3):415–511, 2019. doi:10.1090/bull/1653.

8 Christopher S. Edwards. Some extremal properties of bipartite subgraphs. *Canadian Journal of Mathematics*, 25(3):475–485, 1973. doi:10.4153/CJM-1973-048-x.

9 Christopher S. Edwards. An improved lower bound for the number of edges in a largest bipartite subgraph. In *Proc. 2nd Czechoslovak Symposium on Graph Theory, Prague*, pages 167–181, 1975.

10 Paul Erdős. On some extremal problems in graph theory. *Israel Journal of Mathematics*, 3:113–116, 1965.

11 Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 638–649, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316334.

12 Charles H. Goldberg and Douglas B. West. Bisection of circle colorings. *SIAM Journal on Algebraic Discrete Methods*, 6(1):93–106, 1985. doi:10.1137/0606010.

13 Venkatesan Guruswami. Maximum cut on line and total graphs. *Discrete Applied Mathematics*, 92(2):217–221, 1999. doi:10.1016/S0166-218X(99)00056-6.

14 Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Separations in proof complexity and TFNP. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1150–1161, Los Alamitos, CA, USA, 2022. IEEE Computer Society. doi:10.1109/FOCS54457.2022.00111.

15 Jiří Matoušek. *Lectures on Discrete Geometry*. Graduate Texts in Mathematics. Springer New York, 2002. doi:10.1007/978-1-4613-0039-7.

16 Svatopluk Poljak and Daniel Turzík. A polynomial time heuristic for certain subgraph optimization problems with guaranteed worst case bound. *Discrete Mathematics*, 58(1):99–104, 1986. doi:10.1016/0012-365X(86)90192-5.

17 Sambuddha Roy and William Steiger. Some combinatorial and algorithmic applications of the Borsuk–Ulam theorem. *Graphs and Combinatorics*, 23(1):331–341, June 2007. doi:10.1007/s00373-007-0716-1.

18 William Steiger and Jihui Zhao. Generalized ham-sandwich cuts. *Discrete & Computational Geometry*, 44(3):535–545, 2010. doi:10.1007/s00454-009-9225-8.

19 Arthur H. Stone and John W. Tukey. Generalized "sandwich" theorems. *Duke Math. J.*, 9(2):356–359, 1942. doi:10.1215/S0012-7094-42-00925-6.

Fast Convolutions for Near-Convex Sequences

Cornelius Brand
Institute of Logic and Computation, Vienna University of Technology, Austria

Alexandra Lassota
Max Planck Institute for Informatics, SIC, Saarbrücken, Germany

Abstract

We develop algorithms for $(\min,+)$-Convolution and related convolution problems such as Super Additivity Testing, Convolution 3-Sum and Minimum Consecutive Subsums which use the degree of convexity of the instance as a parameter. Assuming the min-plus conjecture (Künnemann-Paturi-Schneider, ICALP'17 and Cygan et al., ICALP'17), our results interpolate in an optimal manner between fully convex instances, which can be solved in near-linear time using Legendre transformations, and general non-convex sequences, where the trivial quadratic-time algorithm is conjectured to be best possible, up to subpolynomial factors.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases $(\min,+)$-convolution, fine-grained complexity, convex sequences

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.16

Funding The research leading to this article was partially carried out at EPFL, Lausanne, Switzerland, funded by the Swiss National Science Foundation project Complexity of integer Programming (207365).
Cornelius Brand: Supported by FWF grant Y1329 in the START-Program (ParAI).
Alexandra Lassota: Funded by the Swiss National Science Foundation project Complexity of integer Programming (207365).

1 Introduction

The $(\min,+)$-convolution, also called *tropical convolution*, is an operation on sequences that forms the analogue of polynomial multiplication in the tropical semiring: additions are replaced by taking minima, whereas multiplications become additions.

Let $\bar{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$. Formally, the $(\min,+)$-convolution $a * b$ of two sequences $a = (a_0, \ldots, a_n) \in \bar{\mathbb{R}}^{n+1}$ and $b = (b_0, \ldots, b_n) \in \bar{\mathbb{R}}^{n+1}$ is defined as the sequence $c = (c_0, \ldots, c_{2n}) \in \bar{\mathbb{R}}^{2n+1}$ where

$$c_k = \min_{i+j=k} a_i + b_j \text{ for } k = 0, \ldots, 2n. \tag{1}$$

For computational purposes, the inputs are restricted to sequences over $\bar{\mathbb{Z}} = \mathbb{Z} \cup \{\infty\}$. Associated with this operation is the following problem.

$(\min,+)$-Convolution	
Input:	Two sequences $a, b \in \bar{\mathbb{Z}}^{n+1}$
Output:	All entries of $c = a * b$

The trivial algorithm computing $a * b$ given a and b takes a quadratic number of steps in n. Over the years, great efforts have been directed at improving substantially over this trivial bound to no avail. Consequently, it has been conjectured that this is essentially optimal up to subpolynomial factors:

▶ **Conjecture 1** ([19, 25]). *There is no algorithm solving* $(\min,+)$-Convolution *in time* $n^{2-\varepsilon} \cdot \mathrm{polylog}(d)$ *on integral inputs* a, b *of maximal absolute value* d*, for any constant* $\varepsilon > 0$.

While polynomial improvements seem out of reach, there are faster algorithms for the problem, shaving off subpolynomial factors [10, 31, 15].

However, some algorithms can solve the problem in truly subquadratic time, but do so only on restricted input classes. Namely, Chan and Lewenstein [14] give an $O(n^{1.87})$-time algorithm for inputs consisting of monotone sequences with numbers bounded by $O(n)$. This was recently improved to $\tilde{O}(n^{1.5})$ by a break-through result by Chi et al. [16]. Further, there is an algorithm that solves $(\min,+)$-Convolution on *convex* input sequences in time $\tilde{O}(n)$. It can be formulated abstractly as an application of the so-called Legendre transformation, which can even be implemented in linear instead of quasilinear time [27].

Tropical convolutions on general inputs are conjectured to need a least a quadratic running time, whereas there is a linear-time algorithm if both sequences are convex. So far, no results are known which interpolate between fully convex and non-convex sequences in a way that reproduces the quadratic running time in the hardest cases, and the linear running time in the easiest cases. This shows that the role convexity plays is not yet well understood and raises the following question: Can this difference in complexity be quantified? In particular, can the degree of convexity of a sequence be measured in a way that explains both the linear and the quadratic running times in the "most convex" and "most non-convex" cases? We answer these questions in the affirmative. We give two sensible measures to quantify the convexity of an instance. Further, we use them to obtain algorithms which interpolate between convex and non-convex instances.

This line of research can be understood as part of a recent branch of research in parameterized algorithms referred to as FPT-in-P algorithms [2, 22, 23, 24, 28].[1] These are algorithms that admit FPT-style running times of the form $f(k) \cdot n^{O(1)}$, but for problems that are polynomial-time solvable, with the goal of reducing the polynomial dependency on n for small values of k.

Related Problems

Given that $(\min,+)$-Convolution is a well-studied problem within fine-grained complexity theory, there are many other, closely related problems that can either be reduced to $(\min,+)$-Convolution or that $(\min,+)$-Convolution reduces to. The independent articles by Cygan et al. [19] and Künnemann, Paturi and Schneider [25] give an excellent overview over these problems and their relations to each other, and we make no attempt of recalling all of them here.

Yet, there are some specific problems that are relevant in the context of this article. Let us first address some obvious variations of $(\min,+)$-Convolution. Consider the problem obtained from $(\min,+)$-Convolution after changing the min in the definition of $a * b$ for a max, thereby obtaining the following problem.

$(\max,+)$-Convolution	
Input:	Two sequences $a, b \in \bar{\mathbb{Z}}^{n+1}$
Output:	All entries of the sequence $c \in \bar{\mathbb{Z}}^{2n+1}$, where $c_k = \max_{i+j=k} a_i + b_j$

For general inputs, these two cases are interreducible via a pointwise negation of the inputs. Since we are concerned with convexity of inputs, we note that whatever applies to convex sequences for $(\min,+)$-Convolution applies equally to *concave* sequences for $(\max,+)$-Convolution, which are precisely the pointwise negations of convex sequences. In particular,

[1] Of course, as with parameterized algorithms in general, there is a large number of works that fall into the regime of FPT-in-P algorithms before the term had been coined.

our algorithmic results and even our notion of measuring convexity translate in a direct manner to algorithmic results about $(\max, +)$-Convolution. For ease of exposition, we thus only speak about $(\min, +)$-Convolution in the remainder of this article.

Another very closely related problem studied in its maximization-guise in [19] is:

Super Additivity Testing	
Input:	A sequence $a \in \bar{\mathbb{Z}}^{n+1}$
Output:	Whether $a_k \leq \min_{i+j=k} a_i + a_j$ for all k holds

This can be immediately reduced to the special case of $(\min, +)$-Convolution where $a = b$ holds, and $O(n)$ additional comparisons – in particular, all our algorithms for $(\min, +)$-Convolution yield also algorithms for Super Additivity Testing.

One more problem relevant for this article that is close in spirit to $(\min, +)$-Convolution is the following:

Convolution 3-Sum	
Input:	A sequence $t \in \mathbb{Z}^{n+1}$
Output:	Whether there are $0 \leq i \leq j \leq n$ such that $t_i + t_j = t_{i+j}$

This problem is a variant of the well-known 3-Sum problem. Most relevant to our work is the reduction from 3-Sum to Convolution 3-Sum, first appearing as a randomized reduction [29]. Later, this reduction was made deterministic [13].

Finally, we deal with the following problem.

Minimum Consecutive Subsums	
Input:	A sequence $t \in \mathbb{Z}^n$
Output:	For each length $k = 1, \ldots, n$, $\min_{i=1,\ldots,n-k+1} \sum_{j=i}^{i+k-1} t_i$

In prose, we are looking for the minimum consecutive sum of every length in the sequence t. The fine-grained equivalence between $(\min, +)$-Convolution and Minimum Consecutive Subsums was first observed in [26]. Like $(\min, +)$-Convolution, the problem Minimum Consecutive Subsums has an obvious maximization variant Maximum Consecutive Subsums, which we also discuss.

Our Contribution

We present new measures to capture the degree of convexity of an instance of $(\min, +)$-Convolution, namely, the *convex sequence number* and the *convex rank*. We present a linear-time algorithm to compute the convex sequence number of a sequence (and its decomposition into convex sub-sequences). This measure can be large for a particular class of inputs whose convexity can nevertheless be exploited efficiently. To handle such cases, we introduce a generalization to the convex sequence number, which we call the convex rank of a sequence. We prove that we can compute a sufficiently accurate approximation to this parameter and output the corresponding convex sequences in polynomial time.

Secondly, we present efficient algorithms for $(\min, +)$-Convolution, Super Additivity Testing, Convolution 3-Sum, and Minimum Consecutive Subsums under both parameters. This yields $\tilde{O}(sn)$ algorithms for each problem if the input sequence(s) have length n and a convex sequence number of s. As for the convex rank, $(\min, +)$-Convolution, Super Additivity Testing, and Minimum Consecutive Subsums can be solved in time $\tilde{O}(r^2 n)$ for r being the convex rank of the input sequence(s). These algorithms contribute to the understanding of the usefulness of convexity and the structural properties

of instances with bounded convexity. Further, our observations also lead to the result for $(\min,+)$-Convolution on fully convex sequences but avoids the heavy machinery of Legendre transformations.

Related Work

We have already pointed out related algorithmic results on $(\min,+)$-Convolution. More closely related to our bounded-convexity regime, there are other works that consider sequences that are "near-convex" in other ways. For instance, Axiotis-Tzamos [4] consider k-step concave sequences (and the extension to convexity is trivial); Bateni et al. [6] as well as a recent preprint of Bringmann-Cassis [11] use so-called Δ-convexity; furthermore, Arkin et al. define a *convex partition number* [3] in a different context. While all of these measures are well-motivated in the respective papers, they do not meet the criterion that motivates the research reported on in the present paper: they do not allow in a natural manner to interpolate between maximally non-convex sequences (corresponding to parameter value roughly n) and maximally convex (that is, plainly, convex) sequences (corresponding to parameter value 1). It is this gap that this article addresses.

2 Preliminaries

A sequence $a \in \bar{\mathbb{R}}^{n+1}$ is called *convex* if, for all $i = 1, \ldots, n-1$, it holds that

$$2a_i \leq a_{i-1} + a_{i+1}.$$

This can be seen as a discretized version of the characterization that a differentiable function is convex if and only if its derivative is monotonically increasing; in particular, the preceding condition is equivalent to the *slopes* of the sequence satisfying

$$a_i - a_{i-1} \leq a_{i+1} - a_i.$$

By convention, we regard every sequence of length less than three as convex. This is in line with the characterization of convex sequences as those sequences whose piecewise-linear interpolation epigraph is convex.

Let us formally define the matrix consisting of all possible index combinations a and b.

▶ **Definition 2.** *Let* $[a,b] \in \bar{\mathbb{R}}^{(n+1)\times(n+1)}$ *denote the* tropical rank-1 matrix *associated with* $a \in \bar{\mathbb{R}}^{(n+1)}$ *and* $b \in \bar{\mathbb{R}}^{(n+1)}$*, defined through:*

$$[a,b]_{i,j} = a_i + b_j \text{ for } i,j = 0,\ldots,n.$$

▶ Remark 3. There are several inequivalent variants of the tropical rank of a matrix, each stemming from a different characterization of matrix rank in the usual sense. The notion of rank-1 matrices employed here corresponds to what is sometimes called the *Barvinok rank* of a matrix [21].

Define the entries in the k-th antidiagonal for $k = 0,\ldots,2n$ in $[a,b]$ as

$$[a,b]^{(k)} = \begin{cases} ([a,b]_{0,k}, [a,b]_{1,k-1}, \ldots, [a,b]_{k,0}) & \text{if } 0 \leq k \leq n, \\ ([a,b]_{n,k-n}, [a,b]_{n-1,k-n+1}, \ldots, [a,b]_{n-(k-n),2k-2n}) & \text{if } n < k \leq 2n. \end{cases}$$

Computing $a * b$ can be equivalently viewed as finding the minimum in each antidiagonal in $[a,b]_{i,j}$. In particular,

$$c_k = \min [a,b]^{(k)}$$

and thus,

$$a * b = c = (\min\ [a,b]^{(0)}, \ldots, \min\ [a,b]^{(2n)}).$$

The two following lemmas will come in handy for our algorithms.

▶ **Lemma 4.** *Let $a, b \in \bar{\mathbb{R}}^{n+1}$ be convex sequences. Then, for each k, the k-th antidiagonal $[a,b]^{(k)}$ is also a convex sequence.*

Proof. Let $\tilde{a}_k = (a_0, \ldots, a_k)$ for $0 \leq k \leq n$ and $\tilde{a}_k = (a_{k-n}, \ldots, a_n)$ for $n < k \leq 2n$, i.e., the subsequence of a used to form the antidiagonal $[a,b]^{(k)}$. We define analogously $\tilde{b}_k = (b_k, \ldots, b_0)$ for $0 \leq k \leq n$ and $\tilde{b}_k = (b_n, \ldots, b_{k-n})$ for $n < k \leq 2n$. It is clear that $\tilde{a}_k$ is convex for all k. The convexity of $\tilde{b}_k$ for all k follows geometrically from the fact that convexity of the epigraph is retained by reflection along a line parallel to the ordinate. The sum of $\tilde{a}_k$ and $\tilde{b}_k$ is convex since $\tilde{a}_k$ and $\tilde{b}_k$ are, and this sum is precisely $[a,b]^{(k)}$. ◀

▶ **Lemma 5.** *The minimum of a convex sequence $t \in \bar{\mathbb{R}}^{n+1}$ can be computed in time $O(\log n)$.*

Proof. Convex sequences are characterized by the differences between the values at adjacent positions increasing, and their minimum is attained where these differences pass from negative to positive. Since there are n differences and they form a non-decreasing sequence, this can be done in time $O(\log n)$ using binary search. ◀

▶ Remark 6. Note now that Lemmas 5 and 4 imply directly a $\tilde{O}(n)$ algorithm for $(\min, +)$-Convolution on convex sequences, by computing the minima of the $2n+1$ anti-diagonals in time $O(\log n)$. We extend this algorithm based on binary search on the anti-diagonals to more general, non-convex sequences. The same principle will be useful for algorithms for other problems related to $(\min, +)$-convolutions.

In addition to this method, we use a second approach, recently described in [12, 4], that can be employed to design a linear-time algorithm for convolving a convex sequence and an *arbitrary* sequence, convex or not. This is based on a variant of the matrix $[a,b]$, defined as follows:

▶ **Definition 7.** *Pad b with $n+1$ ∞-entries to the left, by setting $b_{-i} = \infty$ for $1 \leq i \leq n+1$, and let $\overline{[a,b]} \in \bar{\mathbb{R}}^{(n+1)\times(n+1)}$ $\overline{[a,b]}_{k,i} = a_i + b_{k-i}$. We call $\overline{[a,b]}$ the* shifted rank-1 matrix *associated to a and b.*

We then make the following observation, as done in [12]:

▶ **Lemma 8.** *Let b be convex. Then $\overline{[a,b]}$ is* Monge*, that is*

$$\overline{[a,b]}_{i,j} + \overline{[a,b]}_{i+1,j+1} \leq \overline{[a,b]}_{i+1,j} + \overline{[a,b]}_{i,j+1}.$$

Proof. First, note that padding b with ∞-entries does not impact convexity. Then, we have that

$$\begin{aligned}
&\overline{[a,b]}_{i,j} + \overline{[a,b]}_{i+1,j+1} - \overline{[a,b]}_{i+1,j} - \overline{[a,b]}_{i,j+1} = \\
&a_i + b_{j-i} + a_{i+1} + b_{j-i} - a_{i+1} - b_{j-i-1} - a_i - b_{j-i+1} = \\
&2b_{j-i} - b_{j-i-1} - b_{j-i+1},
\end{aligned}$$

and $2b_k \leq b_{k-1} + b_{k+1}$ is precisely the definition of b being convex. ◀

▶ Remark 9. Note that instead of searching for minima of anti-diagonals in $[a,b]$, the equivalent task for $\overline{[a,b]}$ is finding the minimum of each row, which, for Monge matrices, can be accomplished in linear time via the SMAWK algorithm [1].

3 Convexity Measures

In the following, we introduce two related, natural definitions for quantifying convexity. Both of these measures come with different advantages and can be used in different settings, witnessing their independent relevance.

The first convexity measure called *convex sequence number* captures the smallest number of cuts to divide a sequence $t \in \bar{\mathbb{R}}^{n+1}$ into convex sub-sequences in a straightforward manner. In turn, the *convex rank* of t is the number of convex sequences of length $n+1$ such that their index-wise minimum equals t. While the convex sequence number is easy to compute, it may be large compared to the convex rank whenever the convex length-$n+1$ sub-sequences are "entangled." We elaborate on an example below. However, we do not know how to compute the convex rank exactly. Instead, we show how to efficiently compute an $O(\log(n))$-approximation algorithm for the convex rank, which is indeed sufficient to (nearly) maintain the running time guarantees of our algorithms.

Convex Sequence Number

Let us define the convex sequence number s formally.

▶ **Definition 10.** *Let $t = (t_0, \ldots, t_n) \in \bar{\mathbb{R}}^{n+1}$ be a sequence. If there exist indices $I = (i_0, i_1 \ldots, i_{s-2})$ with $i_0 < i_1 < \cdots < i_{s-2}$ such that the sequences $(t_0, \ldots, t_{i_0})$, $(t_{i_0+1}, \ldots t_{i_1})$, $\ldots$, $(t_{i_{s-2}+1}, \ldots, t_n)$ are convex and $|I|$ is of minimum cardinality, then we call $s = |I| + 1$ the convex sequence number of t.*

That is, the convex sequence number measures the smallest amount of cuts we have to make in t such that all s sub-sequences are convex, which we believe to be a natural way of measuring how convex a sequence is that is accessible on an intuitive level. We refer to the indices in I as *cuts*. If it is not obvious from the context, we use the convention to denote the convex sequence number of a sequence t by s_t.

Indeed, we can compute the convex sequence number efficiently.

▶ **Theorem 11.** *There is a linear-time algorithm that outputs the minimum number of cuts for a decomposition of a sequence $t \in \bar{\mathbb{R}}^{n+1}$ into convex sub-sequences.*

Proof. We proceed in a greedy fashion. That is, we pass through the sequence and check if each new point added to the sequence satisfies the convexity property.

In detail, we start with the first sub-sequence. Every two next points can always be added (or less, if we already arrived at the end of the instance). For every next new point with index $i+1$, we check whether the second-order difference $t_{i-1} - 2t_i + t_{i+1}$ is non-negative. If so, we add the point and continue. Otherwise, we define $i+1$ as the cut and repeat the procedure.

Regarding minimality, this is indeed optimal as we cannot add any further point to the current sub-sequence and each new point is independent of the points selected before the direct two predecessors. As for correctness, note that the resulting sub-sequences are convex by construction.

It is clear that this procedure takes a linear amount of steps in the input length. ◀

The following property is crucial for our algorithms.

▶ **Lemma 12.** *Let $a, b \in \bar{\mathbb{Z}}^{n+1}$. The parameter convex sequence number is subadditive on each antidiagonal, that is,*

$$s_{[a,b]^{(k)}} \leq s_a + s_b$$

holds for all k.

Proof. Let $I = i_0 < \ldots < i_{s_a-2}$ be the cuts of a decomposition into convex sub-sequences of a, and similarly, $J = j_0 < \cdots < j_{s_b-2}$ for b.

Let $\ell_0 < \ell_1 < \ldots < \ell_{s-2}$ be the union of I and J in increasing order. Note that we can still assume this arrangement to be strictly increasing, since taking the union of two sets removes any potential duplicates. Then, clearly, it holds that $s \leq s_a + s_b$.

It remains to show that $\ell_0, \ldots, \ell_{s-2}$ are indeed the cuts of $[a, b]^{(k)}$ into convex sub-sequences, and thus $s \geq s_{[a,b]^{(k)}}$. Since a and b are per definition convex on each of their sub-sequences, they are also convex on every contiguous subset of them. That is, on each sub-sequence induced by the breakpoints ℓ, a and b are both convex. Hence, $[a, b]^{(k)}$ is convex on these sub-sequences as well, which was to show. ◀

▶ Remark 13. We may at this point already observe the following generalization of the convolution algorithm for purely convex sequences described in Remark 6: Namely, there is an algorithm for $(\min, +)$-Convolution on two sequences $a, b \in \bar{\mathbb{Z}}^{n+1}$ running in time $\tilde{O}((s_a + s_b) \cdot n)$, by employing Theorem 11 and Lemma 12, and observing that the merged index sets in the proof of the latter can be computed in linear time in n.

Convex Rank

For convex sequences, also the procedure from the previous remark can be adapted to use the SMAWK algorithm. To this end, let $b^{(1)}, \ldots, b^{(s_b)} \in \bar{\mathbb{R}}^{n+1}$ be the pieces of the convex sequence decomposition of b, with each $b^{(i)}$ padded with ∞-entries outside the indices corresponding to the i-th index interval in the decomposition of b. Let $c = a * b$ as before. Then, observe that

$$c_k = \min_{s=1,\ldots,s_b} (a * b^{(s)})_k$$

holds for all k, that is, c is given as the point-wise minimum of $a * b^{(1)}, \ldots, a * b^{(s_b)}$. Using the fact that $b^{(s)}$ is convex by definition for each s, we can then apply the SMAWK-based algorithm from Remark 9 to compute c in time $O(\min\{s_a, s_b\} \cdot n)$ This observation can be generalized as follows.

▶ **Lemma 14.** *Let $a, b^{(1)}, \ldots, b^{(r)} \in \bar{\mathbb{R}}^{n+1}$ be any convex sequences, and let $b \in \bar{\mathbb{R}}^{n+1}$ be defined by setting $b_i = \min_\rho b_i^{(\rho)}$, that is, b is the point-wise minimum of the $b^{(\rho)}$. Let $c = a * b$ and $c^{(\rho)} = a * b^{(\rho)}$. Then,*

$$c_k = \min_\rho c_k^{(\rho)}$$

for all k, that is, c is the point-wise minimum of the $c^{(\rho)}$.

Proof. Follows directly from the fact that $x + \min\{y, z\} = \min\{x + y, x + z\}$ for all $x, y, z \in \bar{\mathbb{R}}$. ◀

This observation motivates directly and naturally the following definition:

▶ **Definition 15.** *Let $t = (t_0, \ldots, t_n) \in \bar{\mathbb{R}}^{n+1}$ be a sequence. If there exists a set $T = \{t^{(0)}, t^{(1)} \ldots, t^{(r-1)}\}$ of convex sequences with $t_i \in \bar{\mathbb{R}}^{n+1}$ such that the index-wise minimum satisfies $t_k = \min\{t_k^{(0)}, \ldots, t_k^{(r-1)}\}$, then we call T a* convex r-decomposition *of t. The minimum r such that there exists a convex r-decomposition for t we call the* convex rank *of t.*

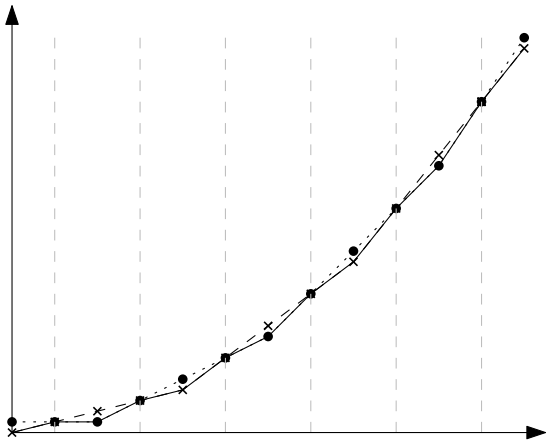


Figure 1 The solid line is the piece-wise linear extension of a non-convex sequence c. The dashed and dotted lines define the piece-wise linear extension of two convex sequences a and b whose point-wise minimum is c (the values of a and b are indicated by disks and crosses, respectively). Dashed, gray vertical lines signify the cut points of the convex sequence number of the sequence c.

Beyond the motivation through the structural observation in Lemma 14, it may require further elaboration to make plausible the supposition that the convex rank is indeed a natural measure of convexity. In essence, it can be viewed as one answer to the question: How many convex sequences are needed to build t from them? Any answer to this question depends on what operation constitutes the formal meaning of "building" a sequence from others.

In the general context of mathematical structures, this is formalized as having a set S with a binary operation $\oplus$ (the "building" operation), such that S contains some distinguished subset $X \subset S$ of particular interest. Then, it is natural to ask for the *X-rank* of an arbitrary element $s \in S$: If $s \in X$, its X-rank is one, and in general it is the minimum number r needed to write $s = x_1 \oplus \cdots \oplus x_r$ with $x_i \in X$ for all $1 \leq i \leq r$. Examples of this abound in the familiar setting of $\oplus$ being ordinary addition over a linear space: If S is a linear space of matrices, X can be taken to be the set of matrices of the form $A = uv^T$ for vectors u and v of the appropriate dimension; in this case, X-rank recovers the ordinary matrix rank. A more involved example from algebraic geometry is furnished by *secant varieties* of a variety X, arising as (the closure of projectivized) ordinary sums of points on the variety X, recovering symmetric, anti-symmetric and tensor (border) rank (when choosing X to be the Veronese, Grassmannian and Segre, respectively), see [8] for ample background on this particular class of examples.

In the more closely related context of tropical convolutions, the operation $\oplus$ being the point-wise minimum of its operands is the natural "additive" operation, forming a semiring together with the operation of tropical convolution. For example, the *Barvinok rank* of a tropical matrix (see [5] and the treatment in [21]), is the smallest number of tropical rank-one matrices needed to express a given matrix as their pointwise minimum. Work by Develin [20] deepens the analogy to ordinary ranks through a tropical analogue of secant varieties. In all of the cases outlined, X-rank corresponds to the complexity of some object with respect to a representation (as a sum) by the set X. Given this mathematical context, it is a very natural thing to measure the degree of convexity through the concept of X-rank of a sequence, where X is the set of convex sequences.

Relation to Convex Sequence Numbers

From its definition, it is clear that the convex rank of a sequence is at most its convex sequence number. However, the two measures can be arbitrarily far apart: Roughly speaking, the critical instances are those where two non-consecutive sub-sequences of t together form a convex sub-sequence (padding the missing indices accordingly). In particular, two convex sequences can be intertwined arbitrarily often. Choosing points which are always at the point-wise minimum of the two sequences, we can produce an arbitrarily large convex sequences number as every time the sequences are intertwined further, their intersection point defines a new cut in the sense of convex sequence numbers. Indeed, consider any prefix of the infinite sequences which are defined for $i \geq 0$ as

$$\begin{aligned} a_{2i} &= i^2 + (i \bmod 2), \\ b_{2i} &= i^2 + 1 - (i \bmod 2), \\ b_{2i+1} = a_{2i+1} &= i^2 + i + 1. \end{aligned}$$

An easy calculation shows that both a and b define convex sequences, but their point-wise minimum has unbounded convex sequence number. See Figure 1 for an illustration.

Algorithms and Properties

An obvious question is how to compute the convex rank of a sequence. First, note the following.

▶ **Proposition 16.** *There is an algorithm running in time $O(r^n)$ to decide whether or not a given sequence $a \in \bar{\mathbb{Z}}^{n+1}$ has convex rank at most r.*

Proof. Let $a \in \bar{\mathbb{Z}}^{n+1}$ and r be given. By definition, for every $i = 0, \ldots, n$ in any convex r-decomposition $a^{(1)}, \ldots, a^{(r)}$ of a there must be some ρ such that $a_i = a_i^{(\rho)}$. Now, we may guess this ρ for every i and obtain r sequences $\hat{a}^{(1)}, \ldots, \hat{a}^{(r)}$ defined partially only at those indices i for which the current guess assumed a given $\hat{a}^{(\rho)}$ to satisfy $a_i = \hat{a}_i^{(\rho)}$. However, it is easy to extend these partial definitions in a piece-wise linear manner (and with ∞ from the left and the right) and check if the resulting sequences are convex and define a as their point-wise minimum. Clearly, if all $\hat{a}^{(\rho)}$ are convex and yield a point-wise, a is of convex rank at most r. If this is not the case for any guess of assignments, then a is of convex rank at least $r + 1$. ◀

In addition to this brute-force approach, we now show how we can approximate the convex rank of a sequence t with a greedy method efficiently, yielding the following theorem. For the remainder of the paper, we write r_t to denote the convex rank of a sequence t.

▶ **Theorem 17.** *There is an $O(\log(n))$-approximation algorithm running in time $O(r_t \cdot n^4 \log n)$ that outputs for a sequence $t \in \bar{\mathbb{R}}^{n+1}$ its decomposition into convex sequences of length $n + 1$ such that their index-wise minimum equals t.*

The algorithm is a greedy algorithm enriched with a dynamic program. The intuition is as follows: We consider the problem as a sort of arithmetic variant of Set Cover. In particular, we are given a sequence of numbers t, and the goal is to "cover" this sequence of numbers with convex sequences $t^{(0)}, \ldots, t^{(r-1)}$, such that for each index i, none of the elements $t_i^{(j)}$ in any sequences j in the covering can be strictly less than t_i. The approach is similar to the greedy approximation algorithm for Set Cover. However, making locally optimal decisions in each step requires a separate dynamic program.

Proof of Theorem 17. We make the assumption that t contains no ∞-entries: trailing and leading ∞-entries can simply be removed from t without changing the optimum. Furthermore, if there is an ∞ in the interior of t, this splits t into two disjoint parts, $t(1)$ and $t(2)$, such that $t = (t(1), \infty, t(2))$. Any convex sequence $t^{(i)}$ that can contribute to a solution (that is, has $t^{(i)}_j = t_j$ for at least one j) can be less than ∞ on at most one of $t(1)$ or $t(2)$, so we may treat $t(1)$ and $t(2)$ as separate instances, and their optimal solutions are the unions of any two optimal solutions of $t(1)$ and $t(2)$, respectively. This procedure can be repeated for any remaining ∞-entries of t.

Now, at each step of our greedy algorithm, we aim to solve the following problem. Given a set of indices $I \subseteq \{0, \dots, n\}$, we wish to compute a convex sequence $t^{(j)} \in \bar{\mathbb{R}}^{n+1}$ that satisfies $t^{(j)}_i \geq t_i$ for all $i \in \{0, \dots, n\}$, and furthermore, meets the following criterion: Let $J_j \subseteq \{0, \dots, n\}$ be the set of indices where equality holds, i.e., $t^{(j)}_i = t_i$ if and only if $i \in J_j$. Then, our goal is to find a sequence $t^{(j)}$ maximizing $|J_j \setminus I|$. The set I will take on the following role in the greedy procedure: At every step, we keep track of the indices in the original sequence t that already have been covered by the selection of sequences up until this point. The set $J_j \setminus I$ is then the set of indices that are covered by the newly constructed sequence $t^{(j)}$.

For this intermediate problem, we now construct a dynamic program that solves it optimally. First, we argue that we can make the following assumption on any optimal solution $t^{(j)}$ without loss of generality: Firstly, the piece-wise linear extension of $t^{(j)}$ is not linear precisely at the indices J_j. Secondly, $t^{(j)}_i = \infty$ for all values of i outside of $[\min J_j, \max J_j]$. Let us refer to such sequences $t^{(j)}$ as *t-compatible.* This can be seen as follows. Consider some convex sequence $t^{(j)}$ that satisfies $t^{(j)}_i \geq t_i$ at all i, and has $t^{(j)}_i = t_i$ at indices $i \in J_j = (j_1, \dots, j_p)$, where $j_1 < \ldots < j_p$. Observe that replacing $t^{(j)}_i$ with ∞ both strictly before and after j_1 and j_p retains convexity of $t^{(j)}$. We further replace the segment of the piece-wise linear extension of $t^{(j)}$ between j_ℓ and $j_{\ell+1}$ with the straight line segment connecting the points $(j_\ell, t^{(j)}_{j_\ell})$ and $(j_{\ell+1}, t^{(j)}_{j_{\ell+1}})$ contained in this epigraph. By the geometric characterization of convexity of a sequence as convexity of the epigraph of its piece-wise linear extension, and in turn by the definition of convexity as containing all line segments between any pair of contained points, the sequence $t^{(j')}$ obtained by this operation has $t^{(j')} \geq t^{(j)} \geq t$ at every point. Moreover, replacing $t^{(j)}$ by $t^{(j')}$ corresponds to intersecting the epigraph of the piece-wise linear extension of $t^{(j)}$ with a collection of half-planes. Therefore, $t^{(j')}$ is a convex sequence satisfying $t^{(j')} \geq t^{(j)} \geq t$ (point-wise), and the restrictions of $t^{(j')}, t^{(j)}$ and t to J_j are all equal. By construction, $t^{(j')}$ is t-compatible.

Consider now Δ, the set of normalized differences between any two (possibly non-consecutive) values of t, that is, $\Delta = \left\{ \frac{t_i - t_j}{|i-j|} \mid i \neq j \right\} \cup \{\pm\infty\}$. In our dynamic program, we keep track of the following data: For each index i and every $\delta \in \Delta$, we are interested in t-compatible sequences $t^{(j)}$ maximizing $|J_j \setminus I|$ among all choices of $t^{(j)}$ such that the following holds: (1) $t^{(j)}_i = t_i$, $t^{(j)}_\ell = \infty$ for all $\ell > i$, and (2) $t^{(j)}_i - t^{(j)}_{i-1} \leq \delta$ if $i > 0$. For each $i > 0$ and δ, let $T[i, \delta]$ contain such a sequence. If indeed these conditions are satisfied, then $\bigcup_i T[i, \infty]$ contains an optimal solution. Now, towards constructing T that contains such an optimal sequence at every index, let first $T[0, \delta] = (t_0, \infty, \dots, \infty)$ for all $\delta > -\infty$, and $T[i, -\infty] = (\infty, \dots, \infty, t_i, \infty, \dots, \infty)$ for all i. Note that the condition $t^{(j)}_i - t^{(j)}_{i-1} \leq -\infty$ forces $t^{(j)}_i < \infty$ and $t^{(j)}_{i-1} = \infty$. Compute then, for every i, the set $V(i)$ of all indices *visible from* i, that is, all indices $j < i$ such that the straight line segment between t_i and t_j is contained in the epigraph of the piece-wise linear extension of t. Then, for $\delta > -\infty$, $T[i+1, \delta]$ is given by the optimum sequence $t^{(j)}$ contained in $\bigcup_{j \in V(i+1)} T[j, \delta]$, extended between j and $i+1$ with the straight-line segment connecting (j, t_j) and (i, t_{i+1}). In particular, $t^{(j)}_{i+1} - t^{(j)}_i = \delta$ in the sequence corresponding to this straight-line extension.

The sequences are t-compatible by construction: The sequence has a possible leading and trailing stretch of ∞, and at all points in between, it is constructed by forming straight-line connections between points of the form (i, t_i) and (j, t_j). Furthermore, the sequences are convex, because the slopes between consecutive line segments are chosen to be non-decreasing. Optimality follows inductively. Now, applying this algorithm greedily yields the desired approximation bound, with an identical analysis as for Set Cover [17].

As for the running time, the table T has $O(n^3)$ entries; computing the sets Δ, as well as $V(i)$ over all i, takes $O(n^2)$ time. Computing a single table entry therefore requires $O(n)$ time because each $V(i)$ is of size $O(n)$, and over all $O(n^3)$ table entries yields an algorithm running time in $O(n^4)$. Performing this greedily at most $O(r_t \log n)$ times gives the claimed running time bound. ◀

▶ Remark 18. The greedy set cover approach has proved useful in a vast number of combinatorial and geometric problems. Notably, e.g. the so-called *convex partition number* [3] admits a similar, if much more direct approximation via the greedy set cover heuristic, which is however known to NP-hard to compute. Another concept related at least in spirit to convex rank is the notion of Barvinok rank, where one asks for the minimum number of tropical rank-one matrices needed to express a given matrix as their entry-wise minimum. This quantity, in turn, is hard to compute (and even approximate) [30]. One pressing question raised by these results is the complexity of exactly computing the convex rank of a sequence.

As it will turn out, the running time for computing the decomposition is the bottleneck in the algorithms. The blow-up for the algorithms itself is negligible though, as it only adds a factor of $O(\log n)$. So, to distinguish between computing the decomposition and the running times of the algorithms, we suppose in the following that the decomposition is given. Such assumptions are commonly used since at least 1993 [18] in the regime of fine-grained complexity and fixed-parameter tractability to distinguish the running times with respect to some parameter and the corresponding computation of the decomposition, see, e.g., the parameters and algorithms for treewidth [9], cliquewidth [18], and twinwidth [7] (which is even NP-hard to compute optimally) among others.

▶ **Lemma 19.** *Let $a, b \in \bar{\mathbb{Z}}^{n+1}$. It holds that*

$$r_{[a,b]^{(k)}} \leq r_a \cdot r_b$$

for all k.

Proof. Denote by $A = \{a^{(0)}, a^{(1)} \ldots, a^{(r_a-1)}\}$ the convex sequences of the decomposition of a, and similarly, $B = \{b^{(0)}, b^{(1)} \ldots, b^{(r_b-1)}\}$ for b.

Set $C = \{a^{(\ell)} * b^{(m)} \mid a^{(\ell)} \in A, b^{(m)} \in B)\}$. We claim that C corresponds to a decomposition of $[a, b]^{(k)}$ into convex sequences.

Let $[a, b]^{(k)} = c = (c_0, \ldots, c_{2n})$ be the entries of the antidiagonal. Each entry c_k is a sum of two sequences $a_i^{(\ell)} \in A$ and $b_{k-i}^{(m)} \in B$ as all entries of a and, respectively, b are preserved in (at least) one of the sequences of the decompositions.

All entries corresponding to one of such combinations $a^{(\ell)}, b^{(m)}$ form a convex sequence as adding two convex sequences remains convex.

It holds that $|C| = r_a \cdot r_b > r_{[a,b]^{(k)}}$. This concludes the proof. ◀

4 Convolution Problems

In this section, we present efficient algorithms for the convolution problems under the paradigm of both convexity measures. Note that $r_a \leq s_a$ implies that, whenever the dependence on r_a in an algorithm is linear, this also implies an algorithm with linear dependence on s_a.

However, there are cases where the algorithms we obtain have running times depending on r_a e.g. quadratically, in which case a separate treatment of parameterizations by s_a still makes sense.

$(\min, +)$-Convolution

We start with the $(\min, +)$-Convolution problem with respect to the convex rank, expediting Lemma 14.

▶ **Theorem 20.** *There is an algorithm for* $(\min, +)$-Convolution *on two sequences* $a, b \in \bar{\mathbb{Z}}^{n+1}$ *running in time* $O(r_a \cdot n)$*, provided a convex rank decomposition* $a^{(1)}, \ldots, a^{(r_a)}$ *of* a *is given.*

Proof. From Lemma 14, it suffices to compute $a^{(\rho)} * b$ for all $\rho = 1, \ldots, r_a$, which can be accomplished in time $O(n)$ using the SMAWK algorithm as pointed out in Remark 9. Taking point-wise minima can be done in time $O(r_a \cdot n)$, and this shows the claimed running time. ◀

▶ Remark 21. This algorithm applies to the case of max-convolution of concave rank decomposition, yielding the same running time bounds. In case the decomposition is not given, we note that the overhead of applying the decomposition algorithm of Theorem 17 to the entire input naively, adding a $O(n^4 r_t)$ preprocessing step, can be ameliorated to some degree: Split a and b into n^c consecutive parts of length n^{1-c} each. Then, preprocess each part separately in time $O(n^{4(1-c)} r_t)$, and use the $O(n^{1-c} r_t)$-time algorithm for each of the n^{2c} pairs of such parts to find the solution of the original instance in time $O(n^{4-3c} + n^{c+1})$, which is minimized for $c = 3/4$, giving an algorithm running in time $O(n^{7/4} r_t)$.

Observe that there is a direct lower bound under the $(\min, +)$-convolution conjecture: There is no algorithm solving $(\min, +)$-Convolution in time less than $((s_a + s_b)n)^{1-\epsilon} \cdot \text{polylog}(d)$ on integral inputs a, b of maximal absolute value d, for any constant $\epsilon > 0$: Suppose that there exists an algorithm solving $(\min, +)$-Convolution in time $((s_a + s_b)n)^{2-\epsilon} \cdot \text{polylog}(d)$ on integral inputs a, b of maximal absolute value d, for any constant $\epsilon > 0$. As each sequence a (and b) can have a convex sequence number of at most $n/2$ (every two consecutive points form one convex sub-sequence), this would mean that we can solve $(\min, +)$-Convolution in time

$$((s_a + s_b)n)^{1-\epsilon} \cdot \text{polylog}(d) \leq ((n/2 + n/2)n)^{1-\epsilon} \leq n^{2-\epsilon'} \cdot \text{polylog}(d)$$

on integral inputs a, b of maximal absolute value d, for some constant $\epsilon' > 0$, contradicting the $(\min, +)$-convolution conjecture.

Minimum Consecutive Subsums

We consider the Minimum Consecutive Subsums problem on convex inputs. Even though there exists a reduction to $(\min, +)$-Convolution, see [26], it does not preserve the convexity of the input. In particular, the reduction computes a such that the i-th entry is the sum of the first i elements in the (in our case convex) input sequence t. For b, the reduction computes the sum of the last i elements as its i-th entry. While the first sequence remains convex, the second becomes concave. Convolving a convex with a concave sequence results in an arbitrary output with respect to the convexity and thus, our algorithm for $(\min, +)$-Convolution cannot be used to obtain an algorithm with the desired running time bound. However, we observe other structural properties of potential solutions, so that we can indeed solve this problem optimally within the same running time bounds as above yielding the following two theorems.

▶ **Theorem 22.** *There is an algorithm for* Minimum Consecutive Subsums *on a sequence* $t \in \bar{\mathbb{Z}}^{n+1}$ *running in time* $\tilde{O}(s_t n)$.

Proof. We first compute the decompositions of t into at most s_t convex sub-sequences in time $O(n)$ using Theorem 11. Next, we divide each convex sub-sequence into two parts (one of which may be empty), namely a monotonically decreasing sub-sequence (preceding the minimum), and a monotonically increasing part (following the minimum). Call the set of monotone sequences $t^{(1)}, \dots, t^{(k)}$ with $k \leq 2s_t$.

When executing the algorithm, we maintain a table M with n entries where the ith entry corresponds to the best solution found so far for a consecutive subsum of length i. Initialize each value with ∞ (or a sufficiently large value). We then sweep once through the whole sequence t and compute for each point t_j the minimum consecutive subsum starting at t_j, but only for all *reasonable* lengths (defined below) $i \in \{1, \dots, n\}$.

More formally, for each point t_j and each monotone sequence $t^{(k)}$, find its reasonable interval $I^{(k)} = [t^{(k)}_\ell, t^{(k)}_{\ell+1}, \dots, t^{(k)}_{\ell_k}]$ such that $t^{(k)}_\ell - t_{j-1} < 0$ (shifting the whole sum one index to the left is not improving) and $t_j - t^{(k)}_{\ell_k+1} < 0$ (shifting the whole sum one index to the right is not improving). Update all entries in the table M which corresponds to the length of consecutive sums from t_j to $t^{(k)}_p$ for all j, k and $t^{(k)}_p \in I^{(k)}$ if they are smaller than the current entry in M. Finally, output M as the solution. During execution, edge cases are simulated by padding with $\pm\infty$ at the end/beginning of monotonically increasing/decreasing sequences.

Correctness follows easily, as each potential interval holding the minimum is considered. Regarding the running time, it is crucial to see that each monotone piece will be divided into non-overlapping intervals by different t_j. Thus, while considering all start points t_j, we overall go through all points once. Hence, computing the reasonable intervals for each t_j and k, and the corresponding consecutive subsum overall costs time $\tilde{O}(s_t n)$ (using a standard trick, we assume that we computed all initial partial sums of t in a single pass, which makes the interval sums $\sum_{i=i_1}^{i_2} t_i$ accessible in time $\tilde{O}(1)$). ◀

▶ **Theorem 23.** *There is an algorithm for* Minimum Consecutive Subsums *on a sequence* $t \in \bar{\mathbb{Z}}^{n+1}$ *running in time* $\tilde{O}(r_t^2 n)$ *given the convex rank decomposition of* t.

Proof. The algorithm proceeds basically as in Theorem 22. However, we now have to deal with multiple convex sequences. This means that for every point t_j, we consider the reasonable intervals of all convex sequences. Again, there will be exactly one interval for each convex sequence and they are non-overlapping. This leads to a blow-up of a factor of r_t, as we now have $r_t n$ instead of n points overall. Also note that once we determined the start index and end index in the interval, we sum up the actual values in t and not the ones from the decomposition into convex sequences. ◀

▶ Remark 24. As we only care about the monotone sub-sequences of the sequence, the concave case can be solved equivalently. Further, by defining the reasonable intervals such that $t^{(k)}_\ell - t_{j-1} > 0$ and $t_j - t^{(k)}_{\ell+1} > 0$, and initializing the table with $-\infty$ entries, we can also solve the maximising version of the problem called Maximum Consecutive Subsums.

Super Additivity Testing

Turning our attention to Super Additivity Testing, we can immediately deduce the following by applying our previous results.

▶ **Theorem 25.** *There is an algorithm for* Super Additivity Testing *on a sequence* $a \in \bar{\mathbb{Z}}^{n+1}$ *running in time* $O(r_a \cdot n)$ *given the convex rank decomposition of* a*.*

Proof. This follows from choosing $b = a$, applying Theorem 20, and then checking whether the result is at least a_i for each i. ◀

▶ Remark 26. As $(\min, +)$-Convolution is used as a subroutine, we can easily see that these algorithms also work for concave inputs.

Convolution 3-Sum

While the previous problems relied on the SMAWK algorithm, we will now employ the strategy laid out in Remarks 6 and 13.

▶ **Theorem 27.** *There is an algorithm for* Convolution 3-Sum *on a sequence* $a \in \bar{\mathbb{Z}}^{n+1}$ *running in time* $\tilde{O}(s_a \cdot n)$*.*

Proof. We first compute the decompositions of a, b into at most s_a convex sub-sequences in time $O(n)$ using Theorem 11. Then, for every anti-diagonal $[a, a]^{(k)}$ of $[a, a]$, the breakpoints of a decomposition of $[a, a]^{(k)}$ into convex sub-sequences is immediate from the proof of Lemma 12. Instead of asking for the minimum (as for $(\min, +)$-Convolution), the Convolution 3-Sum problem asks whether a specific element a_k is contained in the k-th anti-diagonal $[a, a]^{(k)}$ (although finding the minimum of a convex sub-sequence is still part of our algorithm). We can answer this question in $O(\log n)$ time for each of the $O(s_a)$ convex sub-sequences of $[a, a]^{(k)}$, by performing, first, a binary search to find the minimum of each convex sub-sequences, as in the case of $(\min, +)$-Convolution. Each convex sub-sequence is thereby divided into two parts (one of which may well be empty), namely a monotonically decreasing sub-sequence (preceding the minimum), and a monotonically increasing part (following the minimum). On each of those (at most) two parts, we may then perform an ordinary binary search in order to determine the presence or absence of the element a_k in the current convex sub-sequence of $[a, a]^{(k)}$. Correctness and running time follow immediately. ◀

Interestingly, this algorithm cannot be adapted for the parameter convex rank as before. This is due to the issue that we do preserve the real minimum values, but cannot guarantee that each value in the kth antidiagonal corresponds to some sum $t_i + t_j$ for $i + j = k$. Testing this would give an increase in the running time by a factor of n as each antidiagonal could have up to n positions of the desired value, which would be worse than the trivial known algorithm for this problem. It remains an interesting question whether there exists an $O(r_t n)$ algorithm for Convolution 3-Sum.

5 Open Questions

The results in the present paper suggest further research directions: Some of the algorithms could not be shown to be tight as known reduction do not retain the convexity of the instances, see [19]. Also, excluding a running time of the form $O(n + k^2)$ for both convexity measures k would be of great interest. Further, whether or not the quadratic dependency on the convex rank in the algorithms is necessary is yet to be determined. It remains open if we can compute the convex rank exactly in polynomial time or if this problem is NP-hard.

References

1 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.

2 Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004. doi:10.1016/S0196-6774(03)00097-X.

3 Esther M. Arkin, Sándor P. Fekete, Ferran Hurtado, Joseph S. B. Mitchell, Marc Noy, Vera Sacristán, and Saurabh Sethia. On the reflexivity of point sets. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures, 7th International Workshop, WADS 2001, Providence, RI, USA, August 8-10, 2001, Proceedings*, volume 2125 of *Lecture Notes in Computer Science*, pages 192–204. Springer, 2001. doi:10.1007/3-540-44634-6_18.

4 Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster knapsack and graph algorithms. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.19.

5 Alexander I. Barvinok, David S. Johnson, Gerhard J. Woeginger, and Russell Woodroofe. The maximum traveling salesman problem under polyhedral norms. In Robert E. Bixby, E. Andrew Boyd, and Roger Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization, 6th International IPCO Conference, Houston, Texas, USA, June 22-24, 1998, Proceedings*, volume 1412 of *Lecture Notes in Computer Science*, pages 195–201. Springer, 1998. doi:10.1007/3-540-69346-7_15.

6 Mohammad Hossein Bateni, Mohammad Taghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. 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 1269–1282. ACM, 2018. doi:10.1145/3188745.3188876.

7 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is np-complete. In *ICALP*, volume 229 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

8 Alessandra Bernardi, Enrico Carlini, Maria Virginia Catalisano, Alessandro Gimigliano, and Alessandro Oneto. The hitchhiker guide to: Secant varieties and tensor decomposition. *Mathematics*, 6(12), 2018. doi:10.3390/math6120314.

9 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A c^k n 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.

10 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014. doi:10.1007/s00453-012-9734-3.

11 Karl Bringmann and Alejandro Cassis. Faster 0-1-knapsack via near-convex min-plus-convolution. *CoRR*, abs/2305.01593, 2023. doi:10.48550/arXiv.2305.01593.

12 Timothy M. Chan. Approximation schemes for 0-1 knapsack. In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, volume 61 of *OASIcs*, pages 5:1–5:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/OASIcs.SOSA.2018.5.

13 Timothy M. Chan and Qizheng He. Reducing 3sum to convolution-3sum. In Martin Farach-Colton and Inge Li Gørtz, editors, *SOSA*, pages 1–7. SIAM, 2020. doi:10.1137/1.9781611976014.1.

14 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *STOC 2015*, pages 31–40. ACM, 2015. doi:10.1145/2746539.2746568.

15 Timothy M. Chan and R. Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926.

16 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In Stefano Leonardi and Anupam Gupta, editors, *STOC*. ACM, 2022. doi:10.1145/3519935.3520057.

17 V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

18 Bruno Courcelle. Monadic second-order logic and hypergraph orientation. In *LICS*, pages 179–190. IEEE Computer Society, 1993.

19 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min, +)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019. doi:10.1145/3293465.

20 Mike Develin. Tropical secant varieties of linear spaces. *Discrete & Computational Geometry*, 35:117–129, 2006.

21 Mike Develin, Francisco Santos, and Bernd Sturmfels. On the rank of a tropical matrix. *Combinatorial and computational geometry*, 52:213–242, 2005.

22 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theor. Comput. Sci.*, 689:67–95, 2017. doi:10.1016/j.tcs.2017.05.017.

23 Jan M. Hochstein and Karsten Weihe. Maximum s-t-flow with k crossings in $O(k^3 n \log n)$ time. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *SODA*, pages 843–847. SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283473.

24 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for SDD linear systems. In Rafail Ostrovsky, editor, *FOCS*, pages 590–598. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.85.

25 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *ICALP*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.21.

26 Eduardo Sany Laber, Wilfredo Bardales Roncalla, and Ferdinando Cicalese. On lower bounds for the maximum consecutive subsums problem and the (min, +)-convolution. In *IEEE*, pages 1807–1811. IEEE, 2014. doi:10.1109/ISIT.2014.6875145.

27 Yves Lucet. Faster than the fast legendre transform, the linear-time legendre transform. *Numer. Algorithms*, 16(2):171–185, 1997. doi:10.1023/A:1019191114493.

28 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of linear-time data reduction for maximum matching. *Algorithmica*, 82(12):3521–3565, 2020. doi:10.1007/s00453-020-00736-0.

29 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *STOC*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.

30 Yaroslav Shitov. The complexity of tropical matrix factorization. *Advances in Mathematics*, 254:138–156, 2014.

31 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In David B. Shmoys, editor, *STOC*, pages 664–673. ACM, 2014. doi:10.1145/2591796.2591811.

Matrix Completion: Approximating the Minimum Diameter

Diptarka Chakraborty
National University of Singapore, Singapore

Sanjana Dey
National University of Singapore, Singapore

Abstract

In this paper, we focus on the matrix completion problem and aim to minimize the diameter over an arbitrary alphabet. Given a matrix M with missing entries, our objective is to complete the matrix by filling in the missing entries in a way that minimizes the maximum (Hamming) distance between any pair of rows in the completed matrix (also known as the *diameter* of the matrix). It is worth noting that this problem is already known to be NP-hard. Currently, the best-known upper bound is a 4-approximation algorithm derived by applying the triangle inequality together with a well-known 2-approximation algorithm for the radius minimization variant.

In this work, we make the following contributions:

- We present a novel 3-approximation algorithm for the diameter minimization variant of the matrix completion problem. To the best of our knowledge, this is the first approximation result that breaks below the straightforward 4-factor bound.
- Furthermore, we establish that the diameter minimization variant of the matrix completion problem is $(2-\varepsilon)$-inapproximable, for any $\varepsilon > 0$, even when considering a binary alphabet, under the assumption that $\mathsf{P} \neq \mathsf{NP}$. This is the first result that demonstrates a hardness of approximation for this problem.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Incomplete Data, Matrix Completion, Hamming Distance, Diameter Minimization, Approximation Algorithms, Hardness of Approximation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.17

Funding This work was supported by an MoE AcRF Tier 2 grant (MOE-T2EP20221-0009).

1 Introduction

With the advent of big data, the occurrence of missing values in data objects has become increasingly common. These missing entries can arise due to various errors occurring at different stages of data processing, including data collection, data transfer, and data cleaning, and they often even occur arbitrarily. Handling such missing data is widely recognized as a challenging task, and numerous methods, including heuristic, greedy, convex optimization, and statistical approaches, have been proposed in the context of practical applications [1]. One such popular technique is data imputation, which, albeit finds extensive use in data mining, machine learning, and computational biology, requires prior knowledge of the dataset or the adoption of certain statistical assumptions [33].

Addressing the issue of incomplete matrices by filling in missing values is a fundamental problem in data analysis, often approached as an optimization task [10, 20, 21, 29, 16, 17]. In the context of clustering, a popular objective function is to minimize the *cluster diameter* (e.g., [25, 13, 24, 29, 17]), which represents the maximum pairwise distance among data points within a cluster. When dealing with missing entries (or wildcards denoted by $*$), a fundamental question arises: Given an incomplete matrix $M \in (\Sigma \cup \{*\})^{n \times d}$ (over an arbitrary alphabet Σ), how can we fill the $*$-entries with symbols from Σ to obtain a

completion $\bar{M} \in \Sigma^{n \times d}$ that minimizes the maximum pairwise distance between rows? In this paper, we consider the Hamming distance as the underlying distance measure, which is arguably one of the most prevalent distance functions used in a wide range of applications. This problem is known as the Minimum Diameter Matrix Completion (DMC) problem.

The DMC problem is a combinatorial matrix completion problem with numerous applications in coding theory, computational biology, and data science. For instance, in computational biology, the DMC problem arises in assessing the degree of relatedness among genome sequences, where missing entries represent missing data points. The DMC problem is encountered in data science when completing entities with their attributes while satisfying pairwise dissimilarity constraints. The stringology literature extensively explores several consensus problems closely related to DMC [7, 4, 6, 10, 9, 11, 14, 26, 27, 37, 34, 32, 38].

The DMC problem, like most other variants of matrix completion problems, is known to be NP-hard. Koana, Froese, and Niedermeier [29] conducted a comprehensive complexity study on the DMC problem, considering diameter bounds and the maximum number of missing entries, and identified various polynomial-time solvable cases and NP-hard cases. The parameterized complexities of the DMC problem, specifically a more general k-clustering version, have also been investigated in terms of various parameters [16, 17]. Regarding approximation algorithms, only a 4-approximation algorithm is currently known for the DMC problem. This approximation factor is derived from the result of another closely related matrix completion problem called Minimum Radius Matrix Completion (RadMC) [28, 27]. In the RadMC problem, the objective is to find a completion and a "center" string such that the distance between each row of the completed matrix and the center string is minimized. A straightforward application of the triangle inequality shows that any c-approximation (for any $c \geq 1$) to the RadMC problem implies a $2c$-approximation to the DMC problem. Since a simple (folklore) 2-approximation algorithm[1] exists for the RadMC problem, it immediately provides a 4-factor approximate solution for the DMC problem. However, it has been proven that no $(2-\varepsilon)$-approximation algorithm for the RadMC problem exists unless $\mathsf{P} = \mathsf{NP}$ [12], and thus there is no hope of getting a better factor than 4 to the DMC problem by improving the approximation factor of the RadMC problem.

Currently, the possibility of improving the 4-factor approximation for the DMC problem remains an open question. Moreover, there is no known inapproximability result for the DMC problem, leaving room for the plausibility of achieving a polynomial-time approximation scheme (PTAS). In this paper, we refute this possibility by demonstrating that no polynomial-time $(2-\varepsilon)$-approximation algorithm for the DMC problem exists unless $\mathsf{P} = \mathsf{NP}$. Furthermore, we present a 3-approximation algorithm that surpasses the 4-factor bound obtained from a direct application of the triangle inequality, along with a 2-factor algorithm for the RadMC problem.

Our contributions and techniques. One of our primary contributions is a 3-approximation algorithm for the DMC problem, which to the best of our knowledge, is the first one to break below the straightforward 4-approximation bound.

▶ **Theorem 1.** *There is a polynomial-time algorithm that, given an incomplete matrix $M \in (\Sigma \cup \{*\})^{n \times d}$, computes a 3-approximate solution for the Minimum Diameter Matrix Completion (DMC) problem over an arbitrary alphabet Σ.*

[1] The 2-approximation is attained by first solving the LP relaxation of the standard ILP formulation of the RadMC problem and then applying a simple deterministic rounding. We use a similar argument to get a 2-approximation for a restricted version of the DMC problem, namely the DRMC problem (see Appendix A).

To show our result, we consider an intermediate restricted variant of the DMC problem, which we refer to as Minimum Diameter Restricted Matrix Completion (DRMC). In this variant, we add a *column restriction* – all the missing entries of a column must be filled in with the same symbol – for feasible completion of an incomplete matrix. The main advantage of putting this restriction is that now we can formulate this restricted variant as an ILP. Then we solve the corresponding LP relaxation, and finally, applying a simple deterministic rounding, we get a 2-approximation for the DRMC problem (Theorem 5).

Since the only distinction between the DMC and the DRMC problem is the column restriction imposed on feasible completions in the DRMC problem, any feasible solution to the DRMC problem is also a feasible solution to the DMC problem. Surprisingly, we show that for any input incomplete matrix M, the optimum objective value to the DRMC problem is at most 3/2 times that of the DMC problem (Lemma 3). By leveraging this finding along with the 2-approximation algorithm for the DRMC, we can effectively establish Theorem 1.

To build the relationship between the optimal solution of the DRMC and the DMC, we consider an (arbitrary) optimal completion $\bar{M}^*$ for the DMC problem, acknowledging that this completion may not satisfy the column restriction requirement for the DRMC problem. To overcome this, we modify the completion by taking any arbitrary row, say the first row $\bar{M}_1^*$, and then for each column ℓ depending on the symbol at the ℓ-th coordinate of the row $\bar{M}_1^*$ fill in all the missing entries of that column in the whole input (incomplete) matrix. This modification yields a completion $\tilde{M}$ that satisfies the column restriction and thus becomes a feasible solution to the DRMC problem. Let Δ and Δ_R represent the diameters of the completed matrices for DMC ($\bar{M}^*$) and DRMC ($\tilde{M}$), respectively. We aim to demonstrate that $\Delta_R \leq \frac{3}{2} \cdot \Delta$. To provide a high-level idea, let us consider any two rows i, j. It is not hard to see that by applying the triangle inequality, the distance between the rows $\tilde{M}_i$ and $\tilde{M}_j$ is at most twice Δ. However, that only shows $\Delta_R \leq 2\Delta$, which, when combined with a 2-approximation of the DRMC, only gives a 4-approximation to the DMC problem, which is no better than the already known bound. Overcoming this challenge requires developing an argument that surpasses this naive application of the triangle inequality. To tackle this challenge, we divide the rows $\tilde{M}_i$ and $\tilde{M}_j$ into three parts: The first one comprises coordinates where no missing entries are there in both the i-th and j-th row of the input matrix, the second one contains all the coordinates with missing entries in the i-th row but no missing entries in the j-th row, and the third one consists of all the coordinates with missing entries in the j-th row but no missing entries in the i-th row (we disregard the coordinates with missing entries in both the rows because these positions do not contribute to the Hamming distance due to the column restriction). We emphasize that this partitioning is solely for the sake of analysis. Next, we look into these three parts separately and analyze their contributions to the overall Hamming distance. Finally, by using the fact that in $\bar{M}^*$, all the pairwise distances between the rows $\bar{M}_1^*, \bar{M}_i^*$, and $\bar{M}_j^*$ are bounded by Δ, we establish that the distance between the rows $\tilde{M}_i$ and $\tilde{M}_j$ is at most $\frac{3}{2} \cdot \Delta$. The detailed argument is provided in Section 3.

Our next significant contribution is an inapproximability result for the DMC problem. We show that it is NP-hard to get a $(2-\varepsilon)$-approximation, the first inapproximability result for the DMC problem.

▶ **Theorem 2.** *Consider any $\varepsilon > 0$. There is no deterministic polynomial-time algorithm that, given an incomplete matrix $M \in \{0, 1, *\}^{n \times d}$, computes a $(2-\varepsilon)$-approximate solution for the Minimum Diameter Matrix Completion (DMC) problem, unless* P = NP.

We highlight that the aforementioned inapproximability result holds even for a binary alphabet. To establish the inapproximability bound, we employ a reduction from the well-known *Label Cover problem* to a gap version of the DMC problem (Definition 6). Informally

speaking, in the label cover problem, we are presented with a (left and right-regular) bipartite graph with each edge having a function (defined on a label set) as a constraint relation, and the objective is to come up with an assignment (of labels to each vertex) that satisfies "as many" edges as possible (see Definition 7). It is well-known that a gap version of the label cover problem – deciding whether an assignment (of labels) satisfies all the edge constraints or no assignment (of labels) can satisfy more than a small constant fraction of the edges – is NP-hard, even for constant-sized label set and constant left/right-degree bipartite graphs [2, 36]. Given such a label cover instance, we construct a "sparse" incomplete matrix (DMC instance), i.e., a matrix with only a small number of non-$*$ entries per row. For the construction, we utilize the concept of a dictatorship gadget [3] (see Section 4 for the reduction). The completeness of our reduction follows from the properties of the dictatorship gadget. However, for soundness, we need more intricate arguments. The crux of the argument lies in the fact that if the given label cover instance is a No instance (i.e., no assignment can satisfy more than a small constant fraction of the edges), then in our constructed incomplete matrix, for every possible completion we can find "a large" subset of rows where the sum of pairwise distances is large and as a consequence, by averaging a "distant" pair of rows exists.

We remark that a similar proof provides the same inapproximability bound to the restricted version of DMC, namely DRMC, that we consider as an intermediate problem to show our 3-approximation result, establishing that our 2-approximation algorithm for the DRMC is essentially optimal.

Other related works. Various optimization tasks with numerous applications have been investigated in the matrix completion problem [5, 39, 19, 18]. In addition to minimizing the diameter of a cluster, as in the case of the DMC problem, researchers have also studied the problem of radius minimization, known as the Minimum Radius Matrix Completion (RadMC) problem. Alternatively, the RadMC problem is also formulated as the closest string with wildcards problem (or the 1-center in Hamming distance with wildcards). The parameterized complexities of this problem have been explored in [27, 28]. A result of $(2-\varepsilon)$-inapproximability (for any $\varepsilon > 0$) was demonstrated in [12] under the assumption that $\mathsf{P} \neq \mathsf{NP}$, while a 2-approximation algorithm is commonly known. Notably, when there are no missing entries (i.e., without wildcards), the closest string problem admits a polynomial-time approximation scheme (PTAS) [31].

Both the DMC and the RadMC problems are specific variations of clustering problems. In [16, 17], the authors considered a more general version of the clustering problem, where the goal is to partition the rows of an incomplete matrix into clusters while minimizing the diameter or radius of each cluster. Besides radius and diameter, [20] investigated the minimization of rank and the number of distinct rows in the completed matrix. In [21], the authors explore the complexity of completing an incomplete matrix in a way that satisfies specific constraints and can be partitioned into low-rank subspaces. Within the clustering literature, numerous variants of non-combinatorial matrix completion, such as k-center and k-means clustering, have also been extensively studied from the perspective of designing approximation algorithms [22, 23, 30, 35, 15, 8].

2 Preliminaries

Notations. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$. For any $n \times d$ dimensional matrix M, we use M_i to denote the i-th row of M, and $M_i[j]$ (or sometimes for brevity M_{ij}) to denote the (i, j)-th entry of M. Further, for any subset of indices $J = \{j_1, j_2, \ldots, j_k\} \subseteq [d]$, we use $M_i[J]$

to denote the sequence $M_i[j_1]M_i[j_2]\cdots M_i[j_k]$. For two strings $x, y \in \Sigma^d$, we use $\mathcal{H}(x, y)$ to denote the Hamming distance between x and y, which counts the number of coordinates where the symbols of x and y do not match, i.e., $\mathcal{H}(x, y) := |\{i \in [d] \mid x[i] \neq y[i]\}|$.

Matrix Completion. For any incomplete matrix $M \in (\Sigma \cup \{*\})^{n\times d}$, we call $\bar{M} \in \Sigma^{n\times d}$ a valid (feasible) *completion* iff for all $i \in [n]$, $j \in [d]$ with $M_i[j] \neq *$, $\bar{M}_i[j] = M_i[j]$. Sometimes we refer to $\bar{M}$ as a *complete matrix* of M.

Given an incomplete matrix $M \in (\Sigma \cup \{*\})^{n\times d}$, for any feasible completion $\bar{M} \in \Sigma^{n\times d}$ of M, we refer to the quantity $\max_{i\neq j\in[n]} \mathcal{H}(\bar{M}_i, \bar{M}_j)$ as the objective value of $\bar{M}$, denoted by $\texttt{Obj}(\bar{M})$. For any completion $\bar{M}^*$ that minimizes $\texttt{Obj}(\bar{M})$, we denote $\texttt{Obj}(\bar{M}^*)$ by $\mathsf{OPT}_{\text{DMC}}(M)$ (or simply $\mathsf{OPT}(M)$ when the problem DMC is clear from the context). We call a feasible solution $\bar{M}$ a c-approximate solution (for $c \geq 1$) of M iff $\texttt{Obj}(\bar{M}) \leq c \cdot \mathsf{OPT}(M)$.

3 Approximation Algorithm for DMC

In this section, we describe a 3-approximation algorithm for the Minimum Diameter Matrix Completion (DMC) problem over an arbitrary alphabet Σ.

▶ **Theorem 1.** *There is a polynomial-time algorithm that, given an incomplete matrix $M \in (\Sigma \cup \{*\})^{n\times d}$, computes a 3-approximate solution for the Minimum Diameter Matrix Completion (DMC) problem over an arbitrary alphabet Σ.*

In proving the above theorem, we first consider a restricted version of the DMC problem, namely Minimum Diameter Restricted Matrix Completion (DRMC), which, given an incomplete matrix $M \in (\Sigma \cup \{*\})^{n\times d}$, asks to find a (valid) completion $\bar{M}$ of M with a restriction, referred to as *column restriction*, that

- For each column $\ell \in [d]$, for all $i, j \in [n]$, if $M_i[\ell] = M_j[\ell] = *$, then $\bar{M}_i[\ell] = \bar{M}_j[\ell]$,

while minimizing the objective value $\max_{i\neq j\in[n]} \mathcal{H}(\bar{M}_i, \bar{M}_j)$.

It is worth noting that the only difference between DMC and DRMC is that in a complete matrix for DMC, the missing entries of a single column can be completed with different symbols, whereas for DRMC, the missing entries of any particular column must be completed with the same symbol. Thus, it is easy to observe that any feasible solution to the DRMC problem is also a feasible solution to the DMC problem, although the converse may not be true. We provide an LP-based 2-approximation algorithm for the DRMC problem and then argue that it also gives us a 3-approximate solution to the DMC problem. The heart of the argument lies in the relationship between the optimum solution of the DMC problem and that of the DRMC problem.

Relationship between DMC and DRMC. For any incomplete matrix $M \in (\Sigma \cup \{*\})^{n\times d}$, let us use $\mathsf{OPT}_{\text{DMC}}(M)$ and $\mathsf{OPT}_{\text{DRMC}}(M)$ to denote the optimum objective value of the DMC and DRMC problem, respectively. Recall that no matter whether it is the DMC or DRMC problem, the objective value of a (feasible) complete matrix $\bar{M}$ is defined as $\texttt{Obj}(\bar{M}) = \max_{i\neq j\in[n]} \mathcal{H}(\bar{M}_i, \bar{M}_j)$.

▶ **Lemma 3.** *For any $M \in (\Sigma \cup \{*\})^{n\times d}$, $\mathsf{OPT}_{DMC}(M) \leq \mathsf{OPT}_{DRMC}(M) \leq \frac{3}{2} \cdot \mathsf{OPT}_{DMC}(M)$.*

Proof. First, observe that any feasible solution to the DRMC problem is also a feasible solution to the DMC problem. It immediately implies that

$$\mathsf{OPT}_{\text{DMC}}(M) \leq \mathsf{OPT}_{\text{DRMC}}(M).$$

We now focus on proving that $\mathsf{OPT}_{\mathrm{DRMC}}(M) \leq \frac{3}{2} \cdot \mathsf{OPT}_{\mathrm{DMC}}(M)$. For that purpose, let us first consider an (arbitrary) optimal completion $\bar{M}^*$ of M with respect to the DMC problem. Next, we use this solution to come up with a feasible solution (complete matrix) $\tilde{M}$ of M to the DRMC problem. We construct $\tilde{M}$ using $\bar{M}^*$ as follows:

- Consider any arbitrary row, say the first row, of $\bar{M}^*$, i.e., $\bar{M}^*_1$.
- Next, for each $i \in [n]$ and $j \in [d]$, if $M_i[j] = *$, then set $\tilde{M}_i[j] = \bar{M}^*_1[j]$; otherwise set $\tilde{M}_i[j] = M_i[j]$.

It is straightforward to see that $\tilde{M}$ is a feasible completion of M for the DRMC problem. Next, we claim the following.

▷ Claim 4. For all $i, j \in [n]$, $\mathcal{H}(\tilde{M}_i, \tilde{M}_j) \leq \frac{3}{2} \cdot \mathtt{Obj}(\bar{M}^*)$.

This claim is pivotal in proving our lemma. We will prove this claim later, and let us now conclude the proof of the lemma by assuming the above claim.

$$\begin{aligned}\mathtt{Obj}(\tilde{M}) = \max_{i \neq j \in [n]} \mathcal{H}(\tilde{M}_i, \tilde{M}_j) &\leq \frac{3}{2} \cdot \mathtt{Obj}(\bar{M}^*) && \text{(By Claim 4)} \\ &= \frac{3}{2} \cdot \mathsf{OPT}_{\mathrm{DMC}}(M) && \text{(Since } \bar{M}^* \text{ is an optimal solution to DMC).}\end{aligned}$$

Now, since $\tilde{M}$ is a feasible completion of M for the DRMC problem,

$$\mathsf{OPT}_{\mathrm{DRMC}}(M) \leq \mathtt{Obj}(\tilde{M}) \leq \frac{3}{2} \cdot \mathsf{OPT}_{\mathrm{DMC}}(M),$$

which concludes the proof of the lemma. ◀

It now remains to prove Claim 4. Before proceeding with the proof, let us recall that for any subset of indices $J = \{j_1, j_2, \ldots, j_k\} \subseteq [d]$, we use $M_i[J]$ to denote the sequence $M_i[j_1]M_i[j_2] \cdots M_i[j_k]$.

Proof of Claim 4. Consider any $i, j \in [n]$. Let us now consider the indices with $*$-entries in the i-th and the j-th row of the matrix $M \in (\Sigma \cup \{*\})^{n \times d}$. Formally,

$$I := \{\ell \in [d] \mid M_i[\ell] = *\}, \quad J := \{\ell \in [d] \mid M_j[\ell] = *\}, \text{ and } \quad K := [d] \setminus (I \cup J).$$

By the construction of $\tilde{M}$,

$$\tilde{M}_i[I] = \bar{M}^*_1[I], \quad \tilde{M}_j[J] = \bar{M}^*_1[J], \quad \tilde{M}_i[K] = \bar{M}^*_i[K], \quad \tilde{M}_j[K] = \bar{M}^*_j[K]. \tag{1}$$

Let us now focus on the Hamming distances between the subsequences induced by the index sets I, J, and K of the rows $\bar{M}^*_1, \bar{M}^*_i$, and $\bar{M}^*_j$. Let

$$\begin{aligned}
\alpha_1 &:= \mathcal{H}(\bar{M}^*_1[I], \bar{M}^*_i[I]), & \beta_1 &:= \mathcal{H}(\bar{M}^*_1[J], \bar{M}^*_i[J]), & \gamma_1 &:= \mathcal{H}(\bar{M}^*_1[K], \bar{M}^*_i[K]), \\
\alpha_2 &:= \mathcal{H}(\bar{M}^*_1[I], \bar{M}^*_j[I]), & \beta_2 &:= \mathcal{H}(\bar{M}^*_1[J], \bar{M}^*_j[J]), & \gamma_2 &:= \mathcal{H}(\bar{M}^*_1[K], \bar{M}^*_j[K]), \\
\alpha_3 &:= \mathcal{H}(\bar{M}^*_i[I], \bar{M}^*_j[I]), & \beta_3 &:= \mathcal{H}(\bar{M}^*_i[J], \bar{M}^*_j[J]), & \gamma_3 &:= \mathcal{H}(\bar{M}^*_i[K], \bar{M}^*_j[K]).
\end{aligned}$$

See Figure 1 for a pictorial representation of the distances. The Hamming distance between the rows $\tilde{M}_i$ and $\tilde{M}_j$ is

$$\begin{aligned}\mathcal{H}(\tilde{M}_i, \tilde{M}_j) &\leq \mathcal{H}(\tilde{M}_i[I], \tilde{M}_j[I]) + \mathcal{H}(\tilde{M}_i[J], \tilde{M}_j[J]) + \mathcal{H}(\tilde{M}_i[K], \tilde{M}_j[K]) \\ &= \mathcal{H}(\bar{M}^*_1[I], \bar{M}^*_j[I]) + \mathcal{H}(\bar{M}^*_i[J], \bar{M}^*_1[J]) + \mathcal{H}(\bar{M}^*_i[K], \bar{M}^*_j[K]) \quad \text{(By Equation 1)} \\ &= \alpha_2 + \beta_1 + \gamma_3.\end{aligned}$$

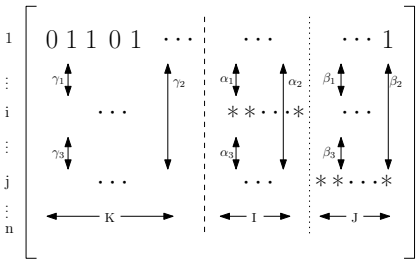


Figure 1 An example matrix M partitioned into coordinate sets I, J, K. For simplicity, in this figure, we assume I and J are disjoint (however, our proof works in full generality).

Thus, to prove our claim, it suffices to argue that

$$\alpha_2 + \beta_1 + \gamma_3 \leq \frac{3}{2} \cdot \texttt{Obj}(\bar{M}^*).$$

Observe that the Hamming distance between the rows $\bar{M}_1^*$ and $\bar{M}_i^*$ is

$$\begin{aligned} &\mathcal{H}(\bar{M}_1^*, \bar{M}_i^*) \leq \alpha_1 + \beta_1 + \gamma_1 \leq \texttt{Obj}(\bar{M}^*) \\ &\implies \alpha_1 + \beta_1 \leq \texttt{Obj}(\bar{M}^*) - \gamma_1. \end{aligned} \tag{2}$$

Similarly, the Hamming distance between the rows $\bar{M}_1^*$ and $\bar{M}_j^*$ is

$$\begin{aligned} &\mathcal{H}(\bar{M}_1^*, \bar{M}_j^*) \leq \alpha_2 + \beta_2 + \gamma_2 \leq \texttt{Obj}(\bar{M}^*) \\ &\implies \alpha_2 + \beta_2 \leq \texttt{Obj}(\bar{M}^*) - \gamma_2. \end{aligned} \tag{3}$$

Also, the Hamming distance between the rows $\bar{M}_i^*$ and $\bar{M}_j^*$ is

$$\begin{aligned} &\mathcal{H}(\bar{M}_i^*, \bar{M}_j^*) \leq \alpha_3 + \beta_3 + \gamma_3 \leq \texttt{Obj}(\bar{M}^*) \\ &\implies \alpha_3 + \beta_3 \leq \texttt{Obj}(\bar{M}^*) - \gamma_3. \end{aligned} \tag{4}$$

Next, the Hamming distance between $\bar{M}_i^*[K]$ and $\bar{M}_j^*[K]$ is

$$\begin{aligned} \gamma_3 &= \mathcal{H}(\bar{M}_i^*[K], \bar{M}_j^*[K]) \\ &\leq \mathcal{H}(\bar{M}_i^*[K], \bar{M}_1^*[K]) + \mathcal{H}(\bar{M}_1^*[K], \bar{M}_j^*[K]) \qquad \text{(By the triangle inequality)} \\ &= \gamma_1 + \gamma_2. \end{aligned} \tag{5}$$

Also, the Hamming distances between $\bar{M}_1^*[I]$ and $\bar{M}_j^*[I]$ is

$$\begin{aligned} \alpha_2 &= \mathcal{H}(\bar{M}_1^*[I], \bar{M}_j^*[I]) \\ &\leq \mathcal{H}(\bar{M}_1^*[I], \bar{M}_i^*[I]) + \mathcal{H}(\bar{M}_i^*[I], \bar{M}_j^*[I]) \qquad \text{(By the triangle inequality)} \\ &= \alpha_1 + \alpha_3 \leq \texttt{Obj}(\bar{M}^*) - \gamma_1 - \beta_1 + \alpha_3 \qquad \text{(By Equation 2)} \end{aligned}$$

which in turn implies that

$$\alpha_2 + \beta_1 \leq \texttt{Obj}(\bar{M}^*) - \gamma_1 + \alpha_3. \tag{6}$$

Similarly, from the Hamming distance between $\bar{M}_i^*[J]$ and $\bar{M}_j^*[J]$, we get the following

$$\beta_1 \leq \beta_2 + \beta_3 \leq \texttt{Obj}(\bar{M}^*) - \gamma_2 - \alpha_2 + \beta_3 \qquad \text{(By Equation 3)}$$

which in turn implies that

$$\alpha_2 + \beta_1 \leq \texttt{Obj}(\bar{M}^*) - \gamma_2 + \beta_3. \tag{7}$$

Adding Equation 6 and Equation 7, we get

$$\begin{aligned} 2(\alpha_2 + \beta_1) &\leq 2 \cdot \texttt{Obj}(\bar{M}^*) - (\gamma_1 + \gamma_2) + (\alpha_3 + \beta_3) \\ &\leq 2 \cdot \texttt{Obj}(\bar{M}^*) - \gamma_3 + (\texttt{Obj}(\bar{M}^*) - \gamma_3) \quad \text{(By Equation 5 and Equation 4)} \\ &\leq 3 \cdot \texttt{Obj}(\bar{M}^*) - 2\gamma_3 \end{aligned}$$

which implies that $\alpha_2 + \beta_1 + \gamma_3 \leq \frac{3}{2} \cdot \texttt{Obj}(\bar{M}^*)$. This completes the proof. ◁

2-approximation for DRMC. In this subsection, we design a 2-approximation algorithm for the DRMC problem, which, when combined with Lemma 3 provides a 3-approximation guarantee for the DMC problem.

▶ **Theorem 5.** *There is a polynomial-time algorithm that, given an incomplete matrix $M \in (\Sigma \cup \{*\})^{n \times d}$, computes a 2-approximate solution for the* Minimum Diameter Restricted Matrix Completion *(DRMC) problem over an arbitrary alphabet Σ.*

We first formulate the problem using an integer linear program (ILP), and then relax the integer constraints to get a linear program (LP), and finally apply a simple (deterministic) rounding scheme on an optimal solution to that LP. We defer the details to Appendix A.

Completing the proof of Theorem 1. Next, we combine Theorem 5 and Lemma 3 to get a 3-approximation algorithm for the DMC problem.

Proof of Theorem 1. Given an incomplete matrix $M \in (\Sigma \cup \{*\})^{n \times d}$, we run the algorithm mentioned in Theorem 5 to get a complete matrix $\bar{M} \in \Sigma^{n \times d}$. Since any feasible solution to the DRMC problem is also a feasible solution to the DMC problem, $\bar{M}$ is a feasible solution to the DMC problem for the input (incomplete) matrix M. Further,

$$\begin{aligned} \texttt{Obj}(\bar{M}) &\leq 2 \cdot \mathsf{OPT}_{\text{DRMC}}(M) && \text{(By Theorem 5)} \\ &\leq 2 \cdot \frac{3}{2} \cdot \mathsf{OPT}_{\text{DMC}}(M) && \text{(By Lemma 3)} \\ &= 3 \cdot \mathsf{OPT}_{\text{DMC}}(M). \end{aligned}$$

Thus $\bar{M}$ is a 3-approximate solution to the DMC problem, which completes the proof. ◀

4 Inapproximability of the DMC problem

In the previous section, we have seen a 3-approximation algorithm for the DMC problem. On the hardness side, so far, we only know that the DMC problem is NP-hard. No inapproximability result is known. In this section, we refute the possibility of getting better than a 2-factor approximation algorithm unless P = NP, even when the alphabet Σ is binary, i.e., $\Sigma = \{0, 1\}$. In particular, we prove Theorem 2.

▶ **Theorem 2.** *Consider any $\varepsilon > 0$. There is no deterministic polynomial-time algorithm that, given an incomplete matrix $M \in \{0, 1, *\}^{n \times d}$, computes a $(2 - \varepsilon)$-approximate solution for the* Minimum Diameter Matrix Completion *(DMC) problem, unless* P = NP.

To show the $(2-\varepsilon)$-inapproximability result, we consider the following gap-version of the DMC problem.

▶ **Definition 6.** *Consider an alphabet Σ and an $\varepsilon > 0$. Given an incomplete matrix $M \in (\Sigma \cup \{*\})^{n \times d}$ and a positive integer g, decide between the following two cases:*

YES: $\mathsf{OPT}(M) \leq g$,
NO: $\mathsf{OPT}(M) > (2-\varepsilon)g$.

Label cover problem and dictatorship gadget. To show the inapproximability result, we provide a reduction from the well-known *label cover* problem to the gap-version of the DMC problem. Let us start by defining the label cover problem.

▶ **Definition 7** (Label Cover Instance). *A* label cover *instance $\Psi = (U, V, E, \Pi)$ consists of*
- *A bipartite graph $G = (U, V, E)$ that is left and right regular. Let D_U and D_V be the degrees of each vertex in U and V respectively,*
- *Label sets $\mathcal{L}_U$ and $\mathcal{L}_V$ for U and V respectively,*
- *For each edge $e \in E$, a function $\pi_e : \mathcal{L}_V \to \mathcal{L}_U$. Let $\Pi = \{\pi_e : \mathcal{L}_V \to \mathcal{L}_U \mid e \in E\}$.*

A labelling *σ is a mapping that assigns each $u \in U$ a label $\sigma(u) \in \mathcal{L}_U$, and each $v \in V$ a label $\sigma(v) \in \mathcal{L}_V$. A labelling σ is said to satisfy an edge $e = (u, v) \in E$ iff $\pi_e(\sigma(v)) = \sigma(u)$. The* value of a labelling *σ, denoted by $\texttt{Val}(\Psi, \sigma)$, is defined as the fraction of edges of E satisfied by σ.*

It is known that a gap version of the label cover problem is NP-hard.

▶ **Theorem 8** ([2, 36]). *For every $\delta \in (0, 1)$, there exists $(1/\delta)^{O(1)}$-sized label sets $\mathcal{L}_U, \mathcal{L}_V$ such that, given a label cover instance $\Psi = (U, V, E, \Pi)$ with label sets $\mathcal{L}_U$ and $\mathcal{L}_V$ and the left degree and the right degree of the instance (bipartite) graph being at most $(1/\delta)^{O(1)}$, it is* NP*-hard to decide between the following two cases:*
- *There exists a labelling σ of Ψ such that $\texttt{Val}(\Psi, \sigma) = 1$,*
- *For every labelling σ of Ψ, $\texttt{Val}(\Psi, \sigma) \leq \delta$.*

One of the standard tools to provide a reduction from the label cover problem is the *dictatorship gadget*. Here, we use a construction of a dictatorship gadget presented in [3]. Before presenting a brief description of the dictatorship construction, let us first introduce a few notions. Let $\neg$ be a negation operator that works both on bits and strings, where the negation of a string is obtained by negating each of its bits individually. A function $f : \{0,1\}^m \to \{0,1\}$ is said to be *odd or folded* if for every x, $f(\neg x) = \neg f(x)$. The oddness of f allows us to store only the value of $f(x)$ for every pair $(x, \neg x)$. If $f(\neg x)$ is needed, we use $\neg f(x)$ instead.

Let us now describe the dictatorship gadget given in [3]. Consider a positive integer k. A *k-dictatorship gadget* is a CNF formula defined over 2^m (for some positive integer m) variables, where an assignment can be viewed as a function $f : \{0,1\}^m \to \{0,1\}$ and assumed to be folded. The set of constraints $\mathcal{C}$ on f is the set of all the clauses of the form $(f(x_1) \vee f(x_2) \vee \cdots \vee f(x_{2k+1}))$, where $x_1, \ldots, x_{2k+1}$ are such that for each $\ell \in [m]$,

$$\sum_{i=1}^{2k+1} x_{i,\ell} \geq k \tag{8}$$

where $x_{i,\ell}$ denotes the ℓ-th bit of the string x_i.

Before proceeding further, let us define a few basic notions. A function $f : \{0,1\}^m \to \{0,1\}$ is said to be a *dictatorship function* if there exists an $\ell \in [m]$ such that for every input $x_i \in \{0,1\}^m$, $f(x_i) = x_{i,\ell}$. For a function $f : \{0,1\}^m \to \{0,1\}$, we call a coordinate $\ell \in [m]$ *relevant* if there exists an input $x_i \in \{0,1\}^m$ such that $f(x_i) \neq f(x_i^{\oplus \ell})$, where $x_i^{\oplus \ell}$ denotes the input obtained by just flipping the ℓ-th bit of x_i. A function f is said to depend on r variables if there are r relevant coordinates. The following result about the dictatorship gadget plays a crucial role in our reduction.

▶ **Lemma 9** ([3]).

1. *If f is a dictatorship function, then it satisfies at least k literals of every clause in the constraint set $\mathcal{C}$.*
2. *Any assignment f that is odd and satisfies all the clauses in the constraint set $\mathcal{C}$ depends on at most $2k-1$ variables.*

It is worth remarking that Item 1 of the above lemma follows immediately from the construction of the dictatorship gadget, especially from Equation 8, whereas Item 2 of the above lemma (which is a weaker converse of Item 1) was shown in [3].

Reduction from the label cover problem. Consider a $\delta \in (0,1)$. Let us consider a label cover instance $\Psi = (U, V, E, \Pi)$, where $\Pi = \{\pi_e : \mathcal{L}_V \to \mathcal{L}_U \mid e \in E\}$. Let us assume that the sizes of both the label set $\mathcal{L}_V$ and $\mathcal{L}_U$ are upper bounded by some $L = (1/\delta)^{\Theta(1)}$. Also, the left degree and the right degree of the instance graph (U, V, E) are upper bounded by some $D = (1/\delta)^{\Theta(1)}$. We associate a function $f_u : \{0,1\}^{|\mathcal{L}_U|} \to \{0,1\}$ (intended to be a dictator of a label of u) to each vertex $u \in U$. Similarly, we associate $f_v : \{0,1\}^{|\mathcal{L}_V|} \to \{0,1\}$ to each $v \in V$. Let us partition the set $\{0,1\}^{|\mathcal{L}_U|}$ into two disjoint equal-sized sets T_U and F_U (arbitrarily) such that for each $x \in T_U$, $\neg x \in F_U$. Similarly, partition the set $\{0,1\}^{|\mathcal{L}_V|}$ into two disjoint equal-sized sets T_V and F_V. (The purpose of this partitioning is that we store the value of the functions only on T_U (and T_V) when the functions are folded.)

Let us consider a positive integer $k = (\min\{L, D, 1/\delta\})^{1/3}$ (which is at most $(1/\delta)^{\Theta(1)}$, and this choice of the value of k is used in the proof of Claim 13). We now construct an incomplete matrix M_Ψ (for brevity, we drop Ψ and simply refer to it as M). For each $u \in U$, consider the k-dictatorship gadget on f_u, and similarly, for each $v \in V$, consider the k-dictatorship gadget on f_v. For each $u \in U$ (resp., $v \in V$), there is a column corresponding to each $x \in T_U$ (resp., $x \in T_V$). (So each column is essentially indexed by either $f_u(x_i)$ for $u \in U$, $x_i \in T_U$, or $f_v(x_i)$ for $v \in V$, $x_i \in T_V$.) Thus, the number of columns is

$$d = |U| \cdot 2^{(|\mathcal{L}_u|-1)} + |V| \cdot 2^{(|\mathcal{L}_v|-1)}.$$

We create rows as follows:

- **Left Vertex Rows:** For each $u \in U$, consider the k-dictatorship gadget on f_u, and let $\mathcal{C}_u$ be the corresponding constraint set. Then add a row for each clause $C \in \mathcal{C}_u$ as follows: For each $x \in T_U$, if the literal represented by $f_u(x)$ is present in C, then set the corresponding entry in the row to be 1; if the literal represented by $f_u(\neg x)$ is present in C, then set the corresponding entry in the row to be 0; otherwise (none of $f_u(x)$ and $f_u(\neg x)$ is present in C), set the corresponding entry in the row to be $*$.
- **Right Vertex Rows:** For each $v \in V$, consider the k-dictatorship gadget on f_v, and let $\mathcal{C}_v$ be the corresponding constraint set. Then add a row for each clause $C \in \mathcal{C}_v$ in a way similar to the above.

- **Edge Rows:** For each edge $e = (u, v) \in E$, add rows as follows: For each possible k inputs $x_1, \ldots, x_k \in \{0,1\}^{|\mathcal{L}_U|}$ on the U side, and $k+1$ inputs $y_1, \ldots, y_{k+1} \in \{0,1\}^{|\mathcal{L}_V|}$ on the V side, we add a row if the following holds:

$$\text{For each label } \ell \in \mathcal{L}_V, \quad \sum_{j=1}^{k} x_{j,\pi_e(\ell)} + \sum_{j=1}^{k+1} y_{j,\ell} \geq k. \tag{9}$$

  In this added row, we set the entries as: If $x_i \in T_U$, then set the entry corresponding to the column $f_u(x_i)$ to be 1; otherwise ($x_i = \neg x'_i$ for some $x'_i \in T_U$), set the entry corresponding to the column $f_u(x'_i)$ to be 0. Similarly, if $y_i \in T_V$, then set the entry corresponding to the column $f_v(y_i)$ to be 1; otherwise ($y_i = \neg y'_i$ for some $y'_i \in T_V$), set the entry corresponding to the column $f_v(y'_i)$ to be 0. All the remaining entries of the row are set to $*$.

It is straightforward to observe that the number of rows n of the constructed matrix M is at most polynomial in the size of the label cover instance (due to our choice of k). Before arguing about the completeness and soundness of the above reduction, let us make a simple observation that immediately follows from the construction of M.

▶ **Observation 10.** *For any label cover instance Ψ, let M be the incomplete matrix constructed as mentioned above. Then each row of M contains exactly $2k+1$ non-$*$ entries.*

Proof. For any left vertex row or right vertex row, by the construction of the k-dictatorship gadget, it contains exactly $2k+1$ non-$*$ entries. For any edge row, by the construction of that row, it contains exactly $2k+1$ non-$*$ entries. ◀

Let us now state the completeness of the reduction, the proof of which is relatively direct from the construction and Lemma 9.

▶ **Lemma 11** (Completeness). *If there exists a labelling σ of Ψ such that $\texttt{Val}(\Psi, \sigma) = 1$, then $\mathsf{OPT}(M) \leq 2k+2$.*

Proof. Let σ be a labeling such that $\texttt{Val}(\Psi, \sigma) = 1$, i.e., for all the edges $e = (u, v) \in E$, $\pi_e(\sigma(v)) = \sigma(u)$. For each $u \in U$, let f_u be the dictatorship function of the label $\sigma(u)$, i.e., for every $x \in \{0,1\}^{|\mathcal{L}_U|}$, $f_u(x)$ is equal to the $\sigma(u)$-th bit of x. Similarly, for each $v \in V$, let f_v be the dictatorship function of the label $\sigma(v)$. Then create a string $s \in \{0,1\}^d$ as follows: For each $u \in U$ and $x \in T_U$ (resp., each $v \in V$ and $x \in T_V$), set the corresponding entry of s to be equal to $f_u(x)$ (resp., $f_v(x)$).

Let us now create a feasible completion $\bar{M}$ by setting each $*$-entry of any column r of M to be $s[r]$ (i.e., the r-th entry of the string s). Next, observe, for each left vertex row or right vertex row M_i, it immediately follows from Item 1 of Lemma 9 that $\mathcal{H}(\bar{M}_i, s) \leq k+1$ (since by Observation 10, there are only $2k+1$ non-$*$ entries in M_i). Also, for each edge row M_i, by the construction (Equation 9), it follows from Observation 10 that $\mathcal{H}(\bar{M}_i, s) \leq k+1$. Hence, for any two $i \neq j$, by the triangle inequality,

$$\mathcal{H}(\bar{M}_i, \bar{M}_j) \leq \mathcal{H}(\bar{M}_i, s) + \mathcal{H}(s, \bar{M}_j) \leq 2k+2.$$ ◀

Next, we consider the more intriguing case of soundness.

▶ **Lemma 12** (Soundness). *For any $\delta \in (0,1)$, there exists an $\varepsilon \in (0,1)$, such that if for every labelling σ of Ψ, $\texttt{Val}(\Psi, \sigma) \leq \delta$, then $\mathsf{OPT}(M) > (2-\varepsilon) \cdot 2k$.*

We devote the rest of this section to proving the soundness.

Proof of soundness. Let us fix a $\delta \in (0,1)$, and set $\varepsilon = (\min\{\delta, 1/k\})^2$ (which is $\delta^{\Theta(1)}$ and this choice of the value of ε is used in the proof of Claim 13). For each row M_i, let us denote the set of coordinates with non-$*$ entries by N_i, i.e.,

$$N_i := \{r \in [d] \mid M_i[r] \neq *\}.$$

We now show the soundness in two steps. First, we argue that if for every labelling σ of Ψ, $\texttt{Val}(\Psi, \sigma) \leq \delta$, then for every plausible completion (represented by a string $s \in \{0,1\}^d$) of any particular row, there exists a "large" subset of rows with every pair of rows having mutually disjoint sets of non-$*$ coordinates. Formally,

▷ Claim 13. If for every labelling σ of Ψ, $\texttt{Val}(\Psi, \sigma) \leq \delta$, then for every row M_p of $M \in (\Sigma \cup \{*\})^{n \times d}$ (where $\Sigma = \{0,1\}$), and for every (feasible) completion $s \in \{0,1\}^d$ of that row, there exists a subset $C_s \subseteq [n]$ of rows of M such that

- $|C_s| \geq 2/\varepsilon + 1$,
- For every $i \in C_s$ and every index $r \in N_i$, $M_i[r] \neq s[r]$,
- For every $i \in C_s$, $N_p \cap N_i = \emptyset$, and
- For every $i \neq j \in C_s$, $N_i \cap N_j = \emptyset$.

Proof. We prove the claim in two parts.

A "large" subset of non-$*$ disjoint rows exists. For a $\delta \in (0,1)$, fix a suitably small constant $\lambda \in (0,1)$ that depends on δ. First, we show that if for every labeling σ of Ψ, $\texttt{Val}(\Psi, \sigma) \leq \delta$, then for every row M_p of M, and for every (feasible) completion $s \in \{0,1\}^d$ of that row, there exists a subset $K_s \subseteq [n]$ of rows of M such that

- $|K_s| \geq (1-\lambda)n$, and
- For every $i \in K_s$ and every index $r \in N_i$, $M_i[r] \neq s[r]$.

The proof of this part resembles the argument used in [3]. We prove the contrapositive of the above statement. For that purpose, let us consider a row M_p, a feasible completion $s \in \{0,1\}^d$ of that row, and a subset $K_s \subseteq [n]$ of size at least $(1-\lambda)n$ such that

$$\text{For every } i \in K_s, \text{ there exists } r \in N_i, M_i[r] = s[r]. \tag{10}$$

Let U' denote the set of all $u \in U$ such that for all the left vertex rows $i \in [n]$ corresponding to u, there exists $r \in N_i$, such that $M_i[r] = s[r]$. Similarly, define $V' \subseteq V$. Also, let E' denote the set of all $e \in E$ such that for all the edge rows $i \in [n]$ corresponding to e, there exists $r \in N_i$, such that $M_i[r] = s[r]$. Recall that the label sets $\mathcal{L}_U$ and $\mathcal{L}_V$ are of size at most $L = (1/\delta)^{O(1)} \leq \mathsf{poly}(k)$ (for our choice of k), and thus the number of left vertex rows (resp., right vertex rows) for each $u \in U$ (resp., $v \in V$) is at most some constant that depends only on k. Also, the left degree and the right degree of the instance graph (U, V, E) are upper bounded by some $D = (1/\delta)^{O(1)} \leq \mathsf{poly}(k)$ (for our choice of k), and thus the number of edge rows is also at most $r(k) \cdot |U|$, where $r(k)$ is some constant that depends only on k. Thus for small enough λ, there exists a constant $\lambda' > 0$ such that

$$|U'| \geq (1-\lambda')|U|, \quad |V'| \geq (1-\lambda')|V|, \text{ and } |E'| \geq (1-\lambda')|E|.$$

Observe, by the construction, for each $u \in U'$ (resp., $v \in V'$), the substring of s corresponding to be positions of f_u (resp., f_v) (viewed as an assignment) satisfies the k-dictatorship gadget for u (resp., v). For simplicity, we refer to these substrings of s as the assignment f_u (resp., f_v). Thus by Item 2 of Lemma 9, f_u (resp., f_v) depends on at most $2k-1$ variables. For each $u \in U'$ (resp., $v \in V'$), let $S_u \subseteq \mathcal{L}_U$ (resp., $S_v \subseteq \mathcal{L}_V$) be the set of variables f_u (resp., f_v) depend on.

Next, we focus on a subset of E', which contains edges with both endpoints in U' and V'. Formally,

$$E'' := \{e = (u, v) \in E' \mid u \in U', v \in V'\}.$$

It is not hard to observe that $|E''| \geq (1 - 3\lambda')|E|$. Furthermore, we claim that for each $e \in E''$, $S_u \cap \pi_e(S_v) \neq \emptyset$. To see this, for the sake of contradiction, assume $S_u \cap \pi_e(S_v) = \emptyset$. Consider k inputs $x_1, \ldots, x_k \in \{0,1\}^{|\mathcal{L}_U|}$ on the U side such that $f_u(x_j) = 0$ and r-th bit of x_j is 1 for all $r \in \mathcal{L}_U \setminus S_u$, and $k+1$ inputs $y_1, \ldots, y_{k+1} \in \{0,1\}^{|\mathcal{L}_V|}$ on the V side such that $f_v(y_j) = 0$ and r-th bit of y_j is 1 for all $r \in \mathcal{L}_V \setminus S_v$. It is easy to verify that this set of inputs satisfies Equation 9. Thus, by the construction, all the bits of the corresponding row in M are different from that of the string s, which is a contradiction.

Now, if we assign labels to each $u \in U'$ and $v \in V'$ by picking labels uniformly at random from S_u and S_v respectively, then each edge $e = (u, v) \in E''$ is satisfied with probability $\frac{1}{|S_u| \cdot |S_v|} \geq \frac{1}{(2k-1)^2}$. This implies that there exists a labeling σ of Ψ such that $\texttt{Val}(\Psi, \sigma) \geq (1 - 3\lambda')/(2k-1)^2 \geq \delta$ (for our choice of k and λ).

A "large" subset of non-$*$ pairwise-disjoint rows exists. Next, let us consider any row M_p of M and any (feasible) completion $s \in \{0,1\}^d$ of that row. We have already argued that there exists a subset $K_s \subseteq [n]$ of rows of M such that

- $|K_s| \geq (1 - \lambda)n$, and
- For every $i \in K_s$ and every index $r \in N_i$, $M_i[r] \neq s[r]$.

We now claim that there exists a subset $C_s \subseteq K_s$ such that

1. $|C_s| \geq 2/\varepsilon + 1$,
2. For every $i \in C_s$ and every index $r \in N_i$, $M_i[r] \neq s[r]$,
3. For every $i \in C_s$, $N_p \cap N_i = \emptyset$, and
4. For every $i \neq j \in C_s$, $N_i \cap N_j = \emptyset$.

We construct a subset $C_s \subseteq K_s$ as follows: Consider the row M_p. If the row M_p is a left/right vertex row for a vertex u, then discard all other left/right vertex rows added for that vertex u, and also all the edge rows corresponding to any of the incident edges of u. If the row M_p is an edge row for an edge $e = (u, v)$, then discard all the other edge rows corresponding to that edge and all the edge rows corresponding to incident edges of u and v, and also all the left/right vertex rows corresponding to u and v. Then, pick a row arbitrarily from the remaining rows from K_s. Again, discard the rows as before and proceed unless we pick $2/\varepsilon + 1$ rows.

Note, $|K_s| \geq (1 - \lambda)n$. Further, recall the number of left vertex rows (resp., right vertex rows) for each $u \in U$ (resp., $v \in V$) is at most some constant that depends only on k, and also the number of edge rows for each edge is at most some constant that depends only on k. Thus, in the above construction of C_s, at each step, we discard at most some constant (that depends only on k) many rows. Hence, the above construction process does not terminate before picking $2/\varepsilon + 1$ rows.

Item 1, 3 and 4 are immediate from the construction. Since $C_s \subseteq K_s$, Item 2 also follows. This concludes the proof of the claim. ◁

Next, we argue that if for every string $s \in \{0,1\}^d$, such a subset C_s exists, then for every feasible completion $\bar{M}$ of M, $\texttt{Obj}(\bar{M}) \geq (2 - \varepsilon) \cdot 2k$.

▷ Claim 14. Let n and d denote the number of rows and columns of M, respectively. If for every row M_p, and any (feasible) completion $s \in \{0,1\}^d$ of that row, there exists a subset $C_s \subseteq [n]$ of rows of M such that

- $|C_s| \geq 2/\varepsilon + 1$,
- For every $i \in C_s$ and every index $r \in N_i$, $M_i[r] \neq s[r]$,
- For every $i \in C_s$, $N_p \cap N_i = \emptyset$, and
- For every $i \neq j \in C_s$, $N_i \cap N_j = \emptyset$.

then $\mathsf{OPT}(M) \geq (2-\varepsilon) \cdot 2k$.

Proof. Let us consider a feasible completion $\bar{M}$ of M, and then consider any arbitrary row, say the first row, of it. Let $s = \bar{M}_1$. Then, by the premise of the claim, there exists a subset $C_s \subseteq [n]$ such that

1. $|C_s| = c \geq 2/\varepsilon + 1$ (the value to be fixed),
2. For every $i \in C_s$ and every index $r \in N_i$, $M_i[r] \neq s[r]$,
3. For every $i \in C_s$, $N_1 \cap N_i = \emptyset$, and
4. For every $i \neq j \in C_s$, $N_i \cap N_j = \emptyset$.

Recall, for any subset of indices $J = \{j_1, j_2, \ldots, j_k\} \subseteq [d]$, we use $\bar{M}_i[J]$ to denote the sequence $\bar{M}_i[j_1]\bar{M}_i[j_2]\cdots\bar{M}_i[j_k]$. For any $i \neq j \in C_s$, let $\alpha_{ij} = \mathcal{H}(\bar{M}_i[N_i], \bar{M}_j[N_i])$ and $\alpha_{ji} = \mathcal{H}(\bar{M}_i[N_j], \bar{M}_j[N_j])$. Thus for any $i \neq j \in C_s$, we have that

$$\mathcal{H}(\bar{M}_i, \bar{M}_j) \geq \mathcal{H}(\bar{M}_i[N_i], \bar{M}_j[N_i]) + \mathcal{H}(\bar{M}_i[N_j], \bar{M}_j[N_j]) = \alpha_{ij} + \alpha_{ji}. \tag{11}$$

Further, consider any $i \neq j \in C_s$. Since $\bar{M}$ is a feasible completion of M, by Item 2, for every index $r \in N_j$, $\bar{M}_j[r] \neq \bar{M}_1[r]$. Now, since $\bar{M} \in \{0,1\}^{n \times d}$, for any index $r \in N_j$, $\bar{M}_i[r] = \bar{M}_1[r]$ if and only if $\bar{M}_i[r] \neq \bar{M}_j[r]$. Thus

$$\mathcal{H}(\bar{M}_1[N_j], \bar{M}_i[N_j]) = |N_j| - \alpha_{ij}. \tag{12}$$

Hence, for any $i \in C_s$, we get that

$$\begin{aligned}
\mathcal{H}(\bar{M}_i, \bar{M}_1) &\geq \sum_{j \in C_s} \mathcal{H}(\bar{M}_i[N_j], \bar{M}_1[N_j]) \\
&= |N_i| + \sum_{j \in C_s : j \neq i} (|N_j| - \alpha_{ij}) && \text{(By Item 2 and Equation 12)} \\
&= (2k+1)c - \sum_{j \in C_s : j \neq i} \alpha_{ij} && \text{(By Observation 10).}
\end{aligned} \tag{13}$$

Now, if for some $i \in C_s$, $\mathcal{H}(\bar{M}_i, \bar{M}_1) \geq 4k$, then clearly $\mathtt{Obj}(\bar{M}) \geq 4k$ and we are done with the proof. So let us assume that for all $i \in C_s$, $\mathcal{H}(\bar{M}_i, \bar{M}_1) \leq 4k$. Then by Equation 13, for every $i \in C_s$,

$$\sum_{j \in C_s : j \neq i} \alpha_{ij} \geq (2k+1)c - 4k = 2k(c-2) + c. \tag{14}$$

Then it follows from Equation 11,

$$\begin{aligned}
\sum_{i \in C_s} \sum_{j \in C_s : j \neq i} \mathcal{H}(\bar{M}_i, \bar{M}_j) &\geq \sum_{i \in C_s} \sum_{j \in C_s : j \neq i} (\alpha_{ij} + \alpha_{ji}) \\
&= 2 \sum_{i \in C_s} \sum_{j \in C_s : j \neq i} \alpha_{ij} \\
&\geq 2c\,(2k(c-2) + c) && \text{(By Equation 14).}
\end{aligned}$$

Then, by a simple averaging, there must exist $i \neq j \in C_s$ such that

$$\mathcal{H}(\bar{M}_i, \bar{M}_j) \geq \frac{2c\,(2k(c-2)+c)}{c(c-1)} > (2-\varepsilon) \cdot 2k$$

where the last inequality follows since $c \geq 2/\varepsilon + 1$. So we have argued that for any feasible completion $\bar{M}$ of M, $\mathtt{Obj}(\bar{M}) > (2-\varepsilon) \cdot 2k$, and hence $\mathsf{OPT}(M) > (2-\varepsilon) \cdot 2k$. ◁

Finally, by combining Claim 13 and Claim 14, we get our desired soundness Lemma 12.

▶ Remark 15. We want to remark that our reduction also establishes $(2-\varepsilon)$-inapproximability for the restricted variant of the DMC problem, namely the Minimum Diameter Restricted Matrix Completion (DRMC) problem, for which we provide a 2-approximation algorithm in Theorem 5. To understand why this is the case, first, observe that we indeed get a solution to the DRMC problem in our completeness proof. For soundness, using a similar (though much simpler) argument that is used in the proof of Lemma 12, we can show that if $\texttt{Val}(\Psi, \sigma) \leq \delta$, then for every string $s \in \{0,1\}^d$, we get at least two rows whose non-$*$ entries do not match with the corresponding entries of s and the set of non-$*$ coordinates are disjoint. Consequently, by an argument similar to that in Claim 14, their distance must be at least $(2-\varepsilon) \cdot 2k$.

5 Conclusion

In this paper, we focus on the task of completing an incomplete matrix while minimizing the diameter, which represents the maximum pairwise distance between any two rows. Currently, the only known approach is a 4-factor approximation algorithm derived from a straightforward utilization of the triangle inequality combined with a simple 2-approximation algorithm for the radius minimization variant. Although the problem is known to be NP-hard, no inapproximability result has been established until now. Our main contribution is the development of a novel 3-approximation algorithm. Notably, this result surpasses the existing 4-factor approximation, marking the first improvement in approximating this problem.

Additionally, we demonstrate that the problem is $(2-\varepsilon)$-inapproximable for any $\varepsilon > 0$, even when considering a binary alphabet. This represents the first inapproximability result for this problem. One of the intriguing open problems is to bridge the gap between the 3-factor approximation and the $(2-\varepsilon)$-inapproximability. Furthermore, it would be interesting to extend our approximation approach to a more general variant of k-clustering, with a focus on minimizing the diameter of each cluster.

References

1 Paul D Allison. *Missing data*. Sage publications, 2001.

2 S Arora, C Lund, R Motwani, M Sudan, and M Szegedy. Proof verification and intractability of approximation problems. In *Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science, IEEE*, 1992.

3 Per Austrin, Venkatesan Guruswami, and Johan Håstad. $(2+\varepsilon)$-SAT is NP-hard. *SIAM Journal on Computing*, 46(5):1554–1573, 2017.

4 Vineet Bafna, Sorin Istrail, Giuseppe Lancia, and Romeo Rizzi. Polynomial and apx-hard cases of the individual haplotyping problem. *Theoretical Computer Science*, 335(1):109–125, 2005.

5 Laura Balzano, Arthur Szlam, Benjamin Recht, and Robert Nowak. K-subspaces with missing data. In *2012 IEEE Statistical Signal Processing Workshop (SSP)*, pages 612–615. IEEE, 2012.

6 Manu Basavaraju, Fahad Panolan, Ashutosh Rai, MS Ramanujan, and Saket Saurabh. On the kernelization complexity of string problems. *Theoretical Computer Science*, 730:21–31, 2018.

7 Christina Boucher, Christine Lo, and Daniel Lokshantov. Consensus patterns (probably) has no eptas. In *Algorithms-ESA 2015: 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 239–250. Springer, 2015.

8 Vladimir Braverman, Shaofeng Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering with missing values. *Advances in Neural Information Processing Systems*, 34:17360–17372, 2021.

9 Laurent Bulteau, Vincent Froese, and Rolf Niedermeier. Tight hardness results for consensus problems on circular strings and time series. *SIAM Journal on Discrete Mathematics*, 34(3):1854–1883, 2020.

10 Laurent Bulteau, Falk Hüffner, Christian Komusiewicz, Rolf Niedermeier, et al. Multivariate algorithmics for NP-hard string problems. *Bulletin of EATCS*, 3(114), 2014.

11 Laurent Bulteau and Markus L Schmid. Consensus strings with small maximum distance and small distance sum. *Algorithmica*, 82(5):1378–1409, 2020.

12 Diptarka Chakraborty, Kshitij Gajjar, and Agastya Vibhuti Jha. Approximating the Center Ranking Under Ulam. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*, volume 213, pages 12:1–12:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

13 Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 1–10, 2001.

14 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Lower bounds for approximation schemes for closest string. *arXiv preprint arXiv:1509.05809*, 2015.

15 Eduard Eiben, Fedor V Fomin, Petr A Golovach, William Lochet, Fahad Panolan, and Kirill Simonov. Eptas for k-means clustering of affine subspaces. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2649–2659. SIAM, 2021.

16 Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. The parameterized complexity of clustering incomplete data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7296–7304, 2021.

17 Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. Finding a cluster in incomplete data. In *30th Annual European Symposium on Algorithms (ESA 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

18 Ehsan Elhamifar. High-rank matrix completion and clustering under self-expressive models. *Advances in Neural Information Processing Systems*, 29, 2016.

19 Ehsan Elhamifar and René Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2765–2781, 2013.

20 Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. Parameterized algorithms for the matrix completion problem. In *International Conference on Machine Learning*, pages 1656–1665. PMLR, 2018.

21 Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. On the parameterized complexity of clustering incomplete data into subspaces of small rank. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3906–3913, 2020.

22 Jie Gao, Michael Langberg, and Leonard J Schulman. Analysis of incomplete data and an intrinsic-dimension helly theorem. *Discrete & Computational Geometry*, 40:537–560, 2008.

23 Jie Gao, Michael Langberg, and Leonard J Schulman. Clustering lines in high-dimensional space: Classification of incomplete data. *ACM Transactions on Algorithms (TALG)*, 7(1):1–26, 2010.

24 Leszek Gasieniec, Jesper Jansson, and Andrzej Lingas. Approximation algorithms for hamming clustering problems. *Journal of Discrete Algorithms*, 2(2):289–301, 2004.

25 Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.

26 Jens Gramm, Rolf Niedermeier, Peter Rossmanith, et al. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37(1):25–42, 2003.

27 Danny Hermelin and Liat Rozenberg. Parameterized complexity analysis for the closest string with wildcards problem. *Theoretical Computer Science*, 600:11–18, 2015.

28 Tomohiro Koana, Vincent Froese, and Rolf Niedermeier. Parameterized algorithms for matrix completion with radius constraints. *arXiv preprint arXiv:2002.00645*, 2020.

29 Tomohiro Koana, Vincent Froese, and Rolf Niedermeier. Binary matrix completion under diameter constraints. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

30 Euiwoong Lee and Leonard J Schulman. Clustering affine subspaces: hardness and algorithms. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 810–827. SIAM, 2013.

31 Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *Journal of the ACM (JACM)*, 49(2):157–171, 2002.

32 Ross Lippert, Russell Schwartz, Giuseppe Lancia, and Sorin Istrail. Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Briefings in bioinformatics*, 3(1):23–31, 2002.

33 Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

34 Christine Lo, Boyko Kakaradov, Daniel Lokshtanov, and Christina Boucher. Seesite: characterizing relationships between splice junctions and splicing enhancers. *IEEE/ACM transactions on computational biology and bioinformatics*, 11(4):648–656, 2014.

35 Yair Marom and Dan Feldman. k-means clustering of lines for big data. *Advances in Neural Information Processing Systems*, 32, 2019.

36 Ran Raz. A parallel repetition theorem. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 447–456, 1995.

37 Markus L Schmid. Finding consensus strings with small length difference between input and solution strings. *ACM Transactions on Computation Theory (TOCT)*, 9(3):1–18, 2017.

38 Lusheng Wang, Ming Li, and Bin Ma. *Closest String and Substring Problems*, pages 321–324. Springer New York, 2016.

39 Jinfeng Yi, Tianbao Yang, Rong Jin, Anil K Jain, and Mehrdad Mahdavi. Robust ensemble clustering by matrix completion. In *2012 IEEE 12th international conference on data mining*, pages 1176–1181. IEEE, 2012.

A Approximation Algorithm for the DRMC Problem

Let us start by restating the theorem.

▶ **Theorem 5.** *There is a polynomial-time algorithm that, given an incomplete matrix* $M \in (\Sigma \cup \{*\})^{n \times d}$*, computes a* 2*-approximate solution for the* Minimum Diameter Restricted Matrix Completion *(DRMC) problem over an arbitrary alphabet* Σ*.*

We now prove the above theorem. We first formulate the problem using an integer linear program (ILP), and then relax the integer constraints to get a linear program (LP), and finally apply a simple (deterministic) rounding scheme on an optimal solution to that LP. It is worth mentioning that the argument used here is very similar to the folklore 2-approximation algorithm for the RadMC problem.

LP relaxation and rounding. Let us first give an ILP formulation. For each column $\ell \in [d]$ and symbol $\sigma \in \Sigma$, we consider a $\{0,1\}$-variable $x_{\ell,\sigma}$. The variable $x_{\ell,\sigma}$ denotes whether all the $*$ entries of M in the ℓ-th column are set to the symbol σ. More specifically, if $x_{\ell,\sigma} = 1$, then all the $*$ entries of M in the ℓ-th column are set to σ.

Let us define δ_{ij} to be the Hamming distance between the non-$*$-entries of the i-th and j-th row of M. More formally, let

$$K_{ij} := \{\ell \in [d] \mid M_i[\ell] \neq * \text{ and } M_j[\ell] \neq *\}\,.$$

Then $\delta_{ij} := \mathcal{H}(M_i[K_{ij}], M_j[K_{ij}])$. For each row $i \in [n]$, let I_i denote the set of indices with non-$*$ entries in M. Formally,

$$I_i := \{\ell \in [d] \mid M_i[\ell] \neq *\}.$$

We use these notations to describe our ILP formulation.

Minimize z

$$s.t. \quad \sum_{\ell \in I_i \setminus K_{ij}} \left(\sum_{\sigma \neq M_i[\ell]} x_{\ell,\sigma} \right) + \sum_{\ell \in I_j \setminus K_{ij}} \left(\sum_{\sigma \neq M_j[\ell]} x_{\ell,\sigma} \right) + \delta_{ij} \leq z \quad \forall i, j \in [n] \tag{15}$$

$$\sum_{\sigma \in \Sigma} x_{\ell,\sigma} = 1 \quad \forall \ell \in [d] \tag{16}$$

$$x_{\ell,\sigma} \in \{0, 1\} \quad \forall \ell \in [d], \forall \sigma \in \Sigma \tag{17}$$

In the above ILP, the constraints 16 ensure that for each column, for all the $*$ entries, exactly one symbol is selected. It is easy to observe that the constraints 15 ask the Hamming distance between i-th and j-th row (for every pair of $i, j \in [n]$) of the output complete matrix to be at most z, which we minimize in the ILP. Hence, the above ILP provides an optimal solution to the DRMC problem on input M.

In order to convert it to LP, we relax the constraints 17 to

$$x_{\ell,\sigma} \in [0, 1], \quad \forall \ell \in [d], \forall \sigma \in \Sigma.$$

Let us consider an optimal solution $\left(x^*_{\ell,\sigma}\right)_{\ell \in [d], \sigma \in \Sigma}$ to the above LP. Next, we use the following simple (deterministic) rounding: For each $\ell \in [d]$, if there exists a symbol $\sigma \in \Sigma$ such that $x^*_{\ell,\sigma} \geq 1/2$ (break ties arbitrarily), then set $\bar{x}_{\ell,\sigma} = 1$, and set $\bar{x}_{\ell,\sigma'} = 0$ for all $\sigma' \neq \sigma$. For an $\ell \in [d]$, if for all $\sigma \in \Sigma$, $x^*_{\ell,\sigma} < 1/2$, then pick a symbol $\sigma \in \Sigma$ arbitrarily and set $\bar{x}_{\ell,\sigma} = 1$, and set $\bar{x}_{\ell,\sigma'} = 0$ for all $\sigma' \neq \sigma$.

It is straightforward to see that by the above rounding, the following holds:

$$\text{For each } \ell \in [d], \quad \sum_{\sigma \in \Sigma} \bar{x}_{\ell,\sigma} = 1.$$

Thus, it provides us with a feasible completion of M for the DRMC problem.

Approximation guarantee. Now we argue that the solution $(\bar{x}_{\ell,\sigma})_{\ell \in [d], \sigma \in \Sigma}$ obtained by the rounding provides a 2-approximate solution to the DRMC problem for the incomplete input matrix M.

Let z^* be the value of z of any optimal solution to our LP formulation. Let $\bar{z}$ be the minimum integer such that

$$\sum_{\ell \in I_i \setminus K_{ij}} \left(\sum_{\sigma \neq M_i[\ell]} \bar{x}_{\ell,\sigma} \right) + \sum_{\ell \in I_j \setminus K_{ij}} \left(\sum_{\sigma \neq M_j[\ell]} \bar{x}_{\ell,\sigma} \right) + \delta_{ij} \leq \bar{z} \qquad \forall i, j \in [n].$$

We want to claim that $\bar{z} \leq 2z^*$, and as a consequence, we get that the solution $\bar{x}_{\ell,\sigma}$ (for all $\ell \in [d], \sigma \in \Sigma$) obtained by the rounding, provides a 2-approximate solution to the DRMC problem.

To show that $\bar{z} \leq 2z^*$, we analyze each term separately in the constraints 15. By our rounding procedure, for any $j \in [n]$, $\ell \in [d]$, $\sum_{\sigma \neq M_j[\ell]} \bar{x}_{\ell,\sigma} = 1$ if and only if $x^*_{\ell,M_j[\ell]} \leq 1/2$ that means $\sum_{\sigma \neq M_j[\ell]} x^*_{\ell,\sigma} \geq 1/2$. Hence, $\sum_{\sigma \neq M_j[\ell]} \bar{x}_{\ell,\sigma} \leq 2 \cdot \sum_{\sigma \neq M_j[\ell]} x^*_{\ell,\sigma}$. Hence,

$$\sum_{\ell \in I_i \setminus K_{ij}} \left(\sum_{\sigma \neq M_i[\ell]} \bar{x}_{\ell,\sigma} \right) + \sum_{\ell \in I_j \setminus K_{ij}} \left(\sum_{\sigma \neq M_j[\ell]} \bar{x}_{\ell,\sigma} \right) + \delta_{ij} \leq 2z \qquad \forall i, j \in [n]$$

which implies $\bar{z} \leq 2z^*$. This concludes the proof of Theorem 5.

Distance Queries over Dynamic Interval Graphs

Jingbang Chen
Cheriton School of Computer Science, University of Waterloo, Canada

Meng He
Faculty of Computer Science, Dalhousie University, Halifax, Canada

J. Ian Munro
Cheriton School of Computer Science, University of Waterloo, Canada

Richard Peng
Cheriton School of Computer Science, University of Waterloo, Canada

Kaiyu Wu
Cheriton School of Computer Science, University of Waterloo, Canada

Daniel J. Zhang
School of Computer Science, Georgia Tech, Atlanta, GA, USA

Abstract

We design the first dynamic distance oracles for interval graphs, which are intersection graphs of a set of intervals on the real line, and for proper interval graphs, which are intersection graphs of a set of intervals in which no interval is properly contained in another.

For proper interval graphs, we design a linear space data structure which supports distance queries (computing the distance between two query vertices) and vertex insertion or deletion in $O(\lg n)$ worst-case time, where n is the number of vertices currently in G. Under incremental (insertion only) or decremental (deletion only) settings, we design linear space data structures that support distance queries in $O(\lg n)$ worst-case time and vertex insertion or deletion in $O(\lg n)$ amortized time, where n is the maximum number of vertices in the graph. Under fully dynamic settings, we design a data structure that represents an interval graph G in $O(n)$ words of space to support distance queries in $O(n \lg n/S(n))$ worst-case time and vertex insertion or deletion in $O(S(n) + \lg n)$ worst-case time, where n is the number of vertices currently in G and $S(n)$ is an arbitrary function that satisfies $S(n) = \Omega(1)$ and $S(n) = O(n)$. This implies an $O(n)$-word solution with $O(\sqrt{n \lg n})$-time support for both distance queries and updates. All four data structures can answer shortest path queries by reporting the vertices in the shortest path between two query vertices in $O(\lg n)$ worst-case time per vertex.

We also study the hardness of supporting distance queries under updates over an intersection graph of 3D axis-aligned line segments, which generalizes our problem to 3D. Finally, we solve the problem of computing the diameter of a dynamic connected interval graph.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Information systems → Data structures

Keywords and phrases interval graph, proper interval graph, intersection graph, geometric intersection graph, distance oracle, distance query, shortest path query, dynamic graph

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.18

Related Version *Thesis Ch. 6*: `https://uwspace.uwaterloo.ca/handle/10012/19686`

1 Introduction

The computation of the shortest path and the distance between a pair of vertices in a graph are fundamental problems in graph algorithms. The *shortest path* between two vertices x and y in an unweighted graph is the path with the fewest number of edges with x and y as endpoints, and the *distance* between x and y is the number of edges in this path.

The study of these problems has many applications including contextual searching [27], social network analysis [28] and molecular topology indices [29].

To support online queries, *shortest path oracles* and *distance oracles* have been designed. They are constructed by preprocessing the given n-vertex graph G, such that, given a pair of vertices x and y of G, the *shortest path query*, which asks for the list of vertices on the shortest path between x and y, or the *distance query*, which asks for the distance between x and y, can be answered efficiently. A naive solution is to precompute information between all pairs of vertices, but the space cost is quadratic. As this space cost is believed to be necessary for fast distance queries, to improve space efficiency, much work has been done to design approximate distance oracles [24, 1]. For example, given a pair of vertices x and y at distance d, the $O(n^{5/3})$-word distance oracles of Pătraşcu and Roditty [24] can compute in constant time an approximate distance in $[d, 2d + 1]$.

These distance oracles often use $O(n^{1+\Omega(1)})$ space, so for modern applications processing large graphs, they tend not to fit in main memory. Therefore, a trend in the design of distance oracles is to take advantage of the structural properties of various classes of graphs to design more space-efficient solutions. The classes of graphs considered include both sparse graphs such as planar graphs [21, 20] and potentially dense graphs such as interval graphs [17, 18] and chordal graphs [25, 23]. Among them, *interval graphs* are intersection graphs of a set of intervals on the real line and have applications in operations research [5] and bioinformatics [30].

The recent result of He et al. [18] is a succinct representation of interval graphs that occupies $n \lg n + (5 + \epsilon)n + o(n)$ bits of space (which is $n + o(n)$ words) for any constant positive ϵ and answers distance queries in constant time. It can also answer shortest path queries using time linear in the length of the path. In addition, they show how to represent a *proper interval graph*, which are interval graphs with no interval properly contained in another, in $2n + o(n)$ bits to provide the same query support.

One reason why the above work on interval graphs is interesting is that, to achieve linear space, it is not possible to store the edges explicitly; unlike planar graphs, the number of edges in an interval graph can possibly be quadratic. Instead, researchers focus on designing data structures over the intervals represented by the graph. Thus, this provides answers to the question of whether one can get more efficient solutions to graph problems when graphs are provided implicitly. In other words, this is an instance in which graphs cannot be written down explicitly, and you want data structures that use linear or near linear space or algorithms that work in linear or near linear time, without explicitly constructing all edges. Other instances include the work of Alman et al. [3] on geometric graphs and that of Munro and Sinnamon [22] on distributive lattices.

Previous work on distance oracles for interval graphs focused on static graphs, while distance oracles for dynamic general graphs require $\Omega(n^2)$ update time [14, 26]. Previous results on dynamic interval graph data structures, worked under the update model of inserting and deleting individual edges, with update times $\Theta(n)$ [13]. Hence, in this paper, we design data structures that support distance and shortest path queries over dynamic interval and proper interval graphs, under the model of updating the graph by the insertion and deletion of an interval, along with all the corresponding edges. By working on these problems, we hope to provide some answers to the following question: "What graphs allow more efficient solutions to dynamic graph problems when the graphs are provided implicitly?"

Previous Work. To solve the all-pairs shortest paths problem over interval graphs, Chen et. al [11] built an $O(n)$-word structure that answers distance queries in $O(1)$ times. Further work by Acan et. al [2] in reducing the space yielded $n \lg n + (3 + \varepsilon)n + o(n)$ (for constant

$\varepsilon > 0$) bits of space data structure for shortest paths, and further improvements of He et al. [18] gave a $n \lg n + (5 + \varepsilon)n + o(n)$ bits of space data structure that also allows distance queries in $O(1)$ time. In the same work, Acan et al. [2] showed how to use $2n + o(n)$ bits of space to support shortest path queries in proper interval graphs in $O(1)$ time per vertex on the path, and He et al.[18] showed how to support distance queries in the same $2n + o(n)$ bits of space, in $O(1)$ time [1].

In a slightly different model of distance labeling, the data structure is distributed among the vertices, and we compute the distance between two vertices from only their labels. The best known result is a $5 \lg n$-bit label of Gavoille and Paul [17] (so a total space of $5n \lg n$ bits), which computes the distance in $O(1)$ time.

Interval graphs are a subset of circular arc graphs, which are intersection graphs of arcs on a circle. The results on interval graphs can be extended to circular arc graphs by unrolling the circular arc graph (twice) and reducing it to an interval graph instance on twice the number of vertices. Interval graphs are also a subset of chordal graphs, which are intersection graphs of subtrees in a tree. For chordal graphs, we have the approximate distance oracle of Singh et. al [25] which uses $O(n)$ words of space and computes in $O(1)$ time a distance between $[d, 2d + 8]$ where d is the true distance. This was improved by Munro and Wu [23] to compute a distance between $[d, d + 1]$ using $n + o(n)$ words. Munro and Wu also gave an exact oracle using $n^2/4 + o(n^2)$ bits of space with query time $O(nf(n))$ for any $f(n) \in \omega(1)$.

Another way of generalizing interval graphs is to define intersection graphs of line segments or boxes in two or higher dimensions. Chan and Skrepetosz [9] solved the all-pairs shortest path problem over several classes of geometric intersection graphs, including intersection graphs of axis-aligned line segments, arbitrary line segments or axis-aligned boxes. Researchers have also designed distance oracles for intersection graphs over unit disks [16, 10].

Our Results. We consider interval graphs given as a set of intervals with edges represented implicitly by the intersections between these intervals. An update is performed by adding a vertex represented by an interval, and this implies the insertions of all its incident edges implicitly, or by deleting vertices represented by an interval which implies the removal of all its incident edges. This is natural in our setting as our edges are implicit in this intersection model, and adding or deleting arbitrary edges may give us a graph outside of our graph class.

Under this model, we design a data structure in Section 3 that represents a proper interval graph G in $O(n)$ words, where n is the number of vertices currently in G, to answer distance queries and to support vertex insertion or deletion in $O(\lg n)$ worst-case time. The shortest path query can also be answered in $O(\lg n)$ worst-case time per vertex on the path.

For general interval graphs, we first consider the incremental case, in which we start from an empty graph and insert n vertices one by one, and the decremental case, in which we start from an n-vertex graph and perform n vertex deletions.

Queries can be made at any time during these update sequences. For these settings, we design in Section 4 an $O(n)$-word representation that supports distance queries in $O(\lg n)$ worst-case time, and vertex insertions (in the incremental case) or deletions (in the decremental case) in $O(\lg n)$ amortized time. The shortest path query can be answered in $O(\lg n)$ worst-case time per vertex on the path.

[1] Although their data structure uses $3n + o(n)$ bits if the graph is not connected, this is due to using n additional bits to determine if vertices belong to the same component. A $O(\sqrt{n})$ additional bit solution to detect components can be found using the equivalence class data structure of El-Zein et al [15].

Under fully dynamic settings in which we can mix insertions and deletions, we further design in Section 5 a data structure that represents a general interval graph G in $O(n)$ words of space to answer distance queries in $O(n \lg n/S(n))$ worst-case time and to support the insertion or deletion of a vertex into or from G in $O(S(n) + \lg n)$ worst-case times, where n is the number of vertices currently in G and $S(n)$ is an arbitrary function that satisfies $S(n) = \Omega(1)$ and $S(n) = O(n)$. It also answers a shortest path query in $O(\lg n)$ worst-case time per vertex on the path. Thus, setting $S(n) = \sqrt{n \lg n}$ yields an $O(n)$-word solution with $O(\sqrt{n \lg n})$-time support for both distance queries and updates. These solutions are the first that support distance and shortest path queries over dynamic interval and proper interval graphs efficiently.

In addition, we also study in Section 6 the hardness of the problem of supporting distance queries under updates in an intersection graph of 3D axis-aligned line segments, which generalizes our problem to 3D. We reduce the online Boolean Matrix vector multiplication (OMv) problem [19] to it. Thus, for any constant $\varepsilon > 0$, the distance query and update times over such a graph cannot be $O(n^{1/2-\varepsilon})$ simultaneously, unless the OMv conjecture [19] is false. This implies conditional lower bounds for more general graphs such as intersection graphs of 3D axis-aligned boxes and intersection graphs of arbitrary line segments in 3D [9, 7, 12, 8].

Due to space constraints, some details are omitted.

2 Definitions and Preliminaries

2.1 Definitions

Let $G = (V, E)$ denote a graph with vertex set V and edge set E, and we consider unweighted graphs only. We use $n = |V|$ and $m = |E|$ to refer to the number of vertices and edges, respectively. As is standard, we assume the word-RAM model with $\Theta(\lg n)$-size words.

An *intersection graph* is formed from a finite family of non-empty sets. We associate each vertex with a set, and two vertices are adjacent if and only if the corresponding sets intersect. Then, an *interval graph* is an intersection graph of a set of intervals on the real line, while a *proper interval graph* is an interval graph where we may associate each vertex with an interval so that no interval is completely covered by another.

The representing interval of a vertex v of an interval graph is denoted by $I_v = [l_v, r_v]$. We use $\mathcal{I}$ to refer to the current interval set of the graph, and we say that $\mathcal{I}$ is an *interval representation* of G. $\mathcal{I}$ may change when the graph is dynamic. We define the following operators over an interval graph G:

- `insert`(v): adds to G a vertex v given by the interval I_v.
- `delete`(v): deletes from G a vertex v given the interval I_v.
- `dist`(u, v): returns the distance between two vertices in G.

2.2 Static Data Structure for Distances in Interval Graphs

Here we will review some previous results for the static case [18, 2], which we will build upon.

For each interval v, we define the parent relationship `parent`(v) as the vertex u such that

$$u = \arg\min \{l_w \mid r_w \geq l_v\} \tag{1}$$

▶ **Definition 1.** *Let G be an interval graph, with a fixed interval representation. The distance tree $T(G)$ is defined under the parent relationship* `parent`(v). *For every vertex v, we order the children of v in order of the left end point of the vertices. That is, if u, w are two children of v with $l_u < l_w$, then u is to the left of w.*

If G is disconnected, then we have a forest instead, with one tree per vertex v where $\texttt{parent}(v) = v$. *Furthermore, in the context of a vertex v, the distance tree $T(G)$ of a disconnected graph G refers to the tree in the forest that contains v.*

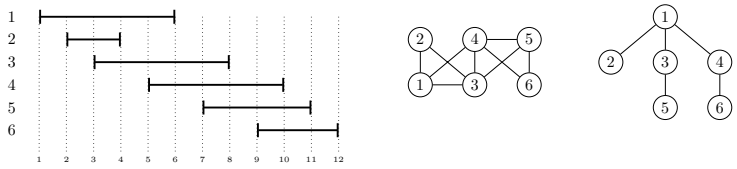


Figure 1 An Interval Graph (middle) with Interval Representation (left), and distance tree constructed (right).

For a tree T and a node v, the quantity $\texttt{pos}_T^X(v)$ denotes the index of v in the X traversal of T, where X could be $level, pre, post$ indicating a breadth-first traversal, preorder traversal and postorder traversal, respectively. We will omit the subscript when the tree being referred to is clear. Then we have

▶ **Lemma 2** (Lemma 7 of [18]). *Let G be a proper interval graph with distance tree $T(G)$ and vertices u, v.* $\texttt{pos}^{level}(u) < \texttt{pos}^{level}(v)$ *if and only if $l_u < l_v$. In the special case that* $\texttt{depth}(u) = \texttt{depth}(v)$, $\texttt{pos}^X(u) < \texttt{pos}^X(v)$ *if and only if $l_u < l_v$ for $X = pre, post, level$.*

The main property of the distance tree is that it encodes distances between vertices.

▶ **Lemma 3.** *Let G be an interval graph with distance tree $T(G)$, and u, v be two vertices in the same connected component of G with $pos^{level}(u) > pos^{level}(v)$. Let the node to root path of u be $u = u_1, \ldots, u_k = r$, and i be the first index where $l_{u_i} \leq r_v$. Then a shortest path from u to v is $u = u_1, \ldots, u_i, v$, and* $\texttt{depth}(u_i)$ *is* $\texttt{depth}(v) - 1$, $\texttt{depth}(v)$ *or* $\texttt{depth}(v) + 1$.

In the proper interval graph case, we may state it more succinctly as

▶ **Lemma 4.** *Let G be a proper interval graph with distance tree $T(G)$ and u, v be two vertices in the same connected component of G with $pos^{level}(u) > pos^{level}(v)$. Then* $\texttt{dist}(u, v) = \texttt{depth}(u) - \texttt{depth}(v) + \mathbf{1}\,(\texttt{pos}^{post}(u) > \texttt{pos}^{post}(v))$, *where the last term evaluates to 1 if the expression inside the brackets is true and 0 otherwise.*

3 Fully Dynamic Proper Interval Graphs

As proper interval graphs are a special class of interval graphs, we naturally modify the $\texttt{insert}(v)$ operation. If the interval $[l, r]$ inserted is incompatible with the proper interval graph - that is it either covers or is covered by another interval, we abort the operation.

Our dynamic solution modifies the distance tree of He et al. [18] so that it is more easily maintainable. The distance tree $T(G)$ (Definition 1) in general has an unbounded degree, which makes updates difficult. To alleviate this, we will apply the well-known isomorphism between ordinal trees and binary trees to ensure that the tree is binary. To preserve depths, sibling edges in the binary tree will have a weight 0, while parent edges in the binary tree will have a weight 1.

More formally, let T be an ordinal tree on n nodes. Define the weighted binary tree T_B also on n nodes as follows: For each vertex v in T_B, the left child of v is its left sibling in T, the right child of v is its rightmost child in T. Left child edges have a weight 0, right child edges have a weight 1. Using this convention, whenever we add a vertex as the left child of

another vertex, it is implicit that we also set the weight of that edge to 0, similarly for right child edges. This transformation preserves post-order traversal. Furthermore, when we talk about node depth or path length in T_B, we refer to weighted node depth or weighted path length, respectively. Hence, any path length in $T(G)$ is invariant under this transformation.

Thus Lemma 4 still holds under this transformation. Furthermore, we will use $\texttt{pos}^{level}(u)$ to refer to the level-order position of u in $T(G)$ before the transformation as the level-order position no longer has any meaning after the transformation. As concepts are more easily stated on $T(G)$, we will mainly use it for stating relationships between vertices, but straightforwardly translate the operations on $T_B(G)$ which we will maintain.

Under this transformation, we also immediately have the analogous notion of ancestors. For a vertex v, the node $u = \texttt{anc}_T(v, d)$ is the ancestor of v at depth d in T. The node u in T_B is the closest ancestor of v at depth d (as the edges may have 0 weights, there are multiple ancestors at each depth). We will store the tree $T_B(G)$ as a top tree [4], which allows us to make dynamic changes along with useful path queries in $O(\lg n)$ time.

▶ **Lemma 5** (Top Tree [4]). *Let T be a forest. A top tree data structure on T occupies $O(n)$ words of space and supports the following operations in $O(\lg n)$ time:* $\texttt{link}(u, v)$*, where u and v are in different trees, links these trees by adding the edge (u, v) to T;* $\texttt{cut}(e)$*, removes the edge e from T;* $\texttt{update\_weight}(e, w)$*, update the weight of the edge e to w;* $\texttt{weighted\_distance}(u, v)$*, returns the weight of the path between u and v;* $\texttt{anc}(u, v, d)$*, returns the first node on the path from u to v at distance d from u.*

For a proper interval graph G, with intervals $\mathcal{I}$, we will maintain the following: **1)**A top tree of $T_B(G)$. For each component, we store a variable indicating the root r of that component. **2)**A mapping between the vertices v of G and the interval $I_v = [l_v, r_v]$. **3)** A mapping from the end points of intervals to the vertex itself. Note that no two intervals can share left end points nor right end points in a proper interval graph, but the left end point of one interval can be the right end point of another. **4)** The left end points of all the intervals. **5)** The right end points of all the intervals. For the last 4 items, we will use a red-black tree, so that searches can be done in $O(\lg n)$ time. For the last 2 items, this also allows us to support successor and predecessor queries, denoted by $\texttt{pred}_L/\texttt{succ}_L$, on the set of left endpoints and $\texttt{pred}_R/\texttt{succ}_R$ on the set of right endpoints. The total space is $O(n)$ words.

Now we translate the distance calculation from $T(G)$ into our data structure essentially by translating the comparison between $\texttt{pos}^{level}$ to a comparison between interval endpoints.

▶ **Lemma 6.** *The data structures in this section can compute $\texttt{dist}(u, v)$ in $O(\lg n)$ time given two vertices, u, v of G.*

Proof. In the case that $\texttt{depth}(u) = \texttt{depth}(v)$, we retrieve the endpoints. Assume without loss of generality that $l_u > l_v$. Since $\texttt{pos}^{post}(u) > \texttt{pos}^{post}(v) \Leftrightarrow \texttt{pos}^{level}(u) > \texttt{pos}^{level}(v) \Leftrightarrow l_u > l_v$, by lemma 4 we have that $\texttt{dist}(u, v) = 1$.

Now suppose that $\texttt{depth}(u) > \texttt{depth}(v)$. By the property of a breadth-first traversal, we also have $\texttt{pos}^{level}(u) > \texttt{pos}^{level}(v)$. Using the top tree, we find $w = \texttt{anc}(u, \texttt{depth}(v))$ in $O(\lg n)$ time. Then $\texttt{pos}^{post}(u) > \texttt{pos}^{post}(v) \Leftrightarrow \texttt{pos}^{post}(w) > \texttt{pos}^{post}(v) \Leftrightarrow l_w > l_v$, so a comparison between l_w and l_v is sufficient to apply lemma 4. ◀

Now we consider the maintaining of the distance tree $T(G)$ (conceptually) and $T_B(G)$ (concretely) under updates. To do so, we first characterize the parent relationship in $T_B(G)$.

▶ **Lemma 7.** *Let G be a proper interval graph with distance tree $T_B(G)$. Let v be a vertex. If v is not a root of one of the components, then* $\texttt{parent}_{T_B(G)}(v)$ *is*

$$\arg\min\left(\{l_w \mid l_w \geq l_v\} \cup \{r_w \mid r_w \geq l_v\}\right) \tag{2}$$

If two vertices u, w have endpoints such that $r_u = l_w$, break ties in the above quantity by treating $l_w < r_u$. Furthermore, let v' be the vertex such that $l_{v'} = \texttt{pred}_L(l_v)$. Then v is the root of its component if and only if v' is not adjacent to v.

Proof. First suppose that v' is adjacent to v. Then, since $l_{v'} < l_v$, we have $r_{v'} \geq l_v$, and hence, v' is a candidate in the parent (in $T(G)$) relationship of v. Therefore, v would have a parent in $T(G)$ and thus would not be the root. Conversely, let p denote the parent of v in $T(G)$. If $v' \neq p$, we have $l_p < l_{v'} < l_v \leq r_p < r_{v'}$. The first and third inequality comes from p being the parent of v, while the others come from the fact that G is a proper interval graph. Thus v' is adjacent to v.

Now suppose that v is not the root of its component. By the construction of $T_B(G)$, v's parent in $T_B(G)$ is either its right sibling or, if it has no right sibling, its parent in $T(G)$. Let $u = \arg\min\{l_w \mid l_w \geq l_v\} \cup \{r_w \mid r_w \geq l_v\}$. Suppose that u is obtained from the first set which consists of left end points. Then as there are no right end points between l_v and l_u, they have the same parent in $T(G)$. Since $l_u = \texttt{succ}_L(l_v)$, u must be the immediate right sibling of v. If u is obtained from the second set which consists of right end points, then as G is a proper interval graph, $u = \arg\min\{l_w \mid r_w \geq l_v\}$, so u is the parent of v in $T(G)$. We note that v is the rightmost child of u in $T(G)$. This can be seen as there are no vertices v' with $l_v \leq l_{v'} \leq r_u$ which any right sibling of v must have. ◀

To support updates, we will first consider insertions. Let w be the vertex with interval $I_w = [l, r]$ be our insertion candidate. We will accomplish this in two steps: first, check that w contains or is contained by some interval in G. If so, we stop immediately. Then, determine the links of $T_B(G)$ that need to be updated. For step 1, we only need to check the containment between the two immediate predecessor and successor of w.

▶ **Lemma 8.** *Let G be a proper interval graph with intervals $\mathcal{I}$. Let $w = [l, r]$ such that $l \neq l_v$ is*[2] *not the left end point of any interval $I_v \in \mathcal{I}$. Let v be the vertex such that $l_v = \texttt{pred}_L(l)$ and u be the vertex such that $l_u = \texttt{succ}_L(l)$.*

Then w is contained in some interval $I_{v'}$ if and only if w is contained in I_v, and w contains some interval $I_{u'}$ if and only if w contains I_u.

Proof. By assumption, $l_v \neq l_w \neq l_u$. As v is the predecessor of w, $l_{v'} \leq l_v$. If w is contained in $I_{v'}$, then $l_{v'} \leq l_v < l_w < r_w \leq r_{v'} < r_v$, so w is also contained in I_v. Conversely, we choose $v' = v$. Now suppose that w contains some interval $I_{u'}$. Then we have $l_w < l_u \leq l_{u'} \leq r_u < r_{u'} < r_w$, so w contains u. ◀

We will now assume that $G \cup \{w\}$ is a proper interval graph.

▶ **Lemma 9.** *$O(1)$ links need to be updated to transform $T_B(G)$ to $T_B(G \cup \{w\})$.*

Proof. By Lemma 7 for a vertex v, the parent in $T_B(G)$ is given by equation 2. By adding l_w and r_w, we may need to add an edge between w and $\texttt{parent}_{T_B(G \cup \{w\})}(w)$, and we also add links whenever w is the result of equation 2. Furthermore, as roots do not have parents, if w becomes the new root of some component, the old root would need to relink as well.

[2] If $l = l_v$, then one of w and v contain the other, and we can easily handle this case by aborting.

Thus by the analysis above, at most 4 links need to be updated. We note that to compute the new parent of any node, equation 2 can be calculated using $\texttt{succ}_L(l_v)$ and $\texttt{succ}_R(l_v)$, then taking the minimum of the result.

To be complete, we will explicitly state the vertices that need to be relinked. If w is the new root of some component, then the old root is $l_r = \texttt{succ}_L(l_w)$ if r is adjacent to w. Otherwise, w is in a component by itself. In the case that r is adjacent to w, r will need to recalculate its parent link. If w is not the new root of some component, then we calculate the parent of w. The two children of w are the vertices whose left end points immediately precede l_w and r_w. Let u, v be the vertices such that $l_u = \texttt{pred}_L(l_w)$ and $l_v = \texttt{pred}_L(r_w)$. We may need to relink u and v as they may now be children of w. ◀

▶ **Lemma 10.** *The* `insert` *operation has time complexity* $O(\lg n)$.

Proof of Lemma 10. `insert` first checks that w is consistent with the other intervals in $O(\lg n)$ time. The transformation from $T_B(G)$ to $T_B(G \cup \{w\})$ requires the relinking of $O(1)$ links, which takes $O(\lg n)$ time. Adding w to the maps between vertices and end points and adding the end points of w to the trees require $O(\lg n)$ time. Thus in total, `insert` requires $O(\lg n)$ time. ◀

The `delete` operation is in essence the reverse of `insert`. Details for it and the following query, used in Section 4, are omitted. Given two vertices u, v (represented by intervals) not in G, compute $\texttt{dist}(u, v)$ in $G \cup \{u, v\}$ without requiring $G \cup \{u, v\}$ to be a proper interval graph. The main idea is that if $l_u < r_u < l_v$, then $\texttt{dist}(u, v)$ depends only on r_u and l_v. Thus we compute two vertices u', v' with $r_u = r_{u'}, l_v = l_{v'}$ so that $G \cup \{u', v'\}$ is a proper interval graph, and the distance can be computed by replacing u, v with u', v'. Thus we have:

▶ **Theorem 11.** *A proper interval graph* G *can be represented in* $O(n)$ *words of space, where* n *is the number of vertices currently in* G*, to support* `insert`, `delete`, `dist` *in* $O(\lg n)$ *time, and* `shortest_path` *in* $O(\lg n)$ *time per vertex on the path. Furthermore,* `dist` *supports arguments not in the graph* G*: for intervals* $x = [l_x, r_x], y = [l_y, r_y]$ *not necessarily in* G, $\texttt{dist}_{G \cup \{x,y\}}(x, y)$ *is supported in* $O(\lg n)$ *time.*

4 Dynamic Interval Graphs in Incremental and Decremental Settings

In this section, we will study dynamic interval graphs in the incremental and decremental settings; Some details are omitted.

We do this by observing that any interval that is contained in some other interval can be removed without changing the length of the shortest paths. By maintaining the set of remaining intervals, which we will say are *exposed*, we reduce the problem to the fully dynamic proper case. In the incremental setting, once an interval becomes contained by another interval (that is, no longer exposed), it will remain so for the remaining operations; in the decremental setting, once an interval is no longer contained by another interval (that is, becomes exposed), it will remain so until it is deleted. Hence, the total number of updates to the proper interval graph data structure will be $O(n)$. The two main theorems for this section are:

▶ **Theorem 12** (Incremental)**.** *There is a data structure that maintains an interval graph* G *in* $O(n)$ *words of space, where* n *is the total number of vertices to be inserted into* G*, and supports* `dist` *in* $O(\lg n)$ *worst-case time and* `insert` *in* $O(\lg n)$ *amortized time.*

▶ **Theorem 13** (Decremental). *There is a data structure that starts with a given n-vertex interval graph G and uses $O(n)$ words to support* `dist` *in $O(\lg n)$ worst-case time and* `delete` *in $O(\lg n)$ amortized time.*

As we briefly explained, these will be solved (in an amortized fashion) by reducing to the problem on dynamic proper interval graphs. Formally, for an interval graph G with intervals $\mathcal{I}$, we say that an interval $x \in \mathcal{I}$ is exposed if it is not contained by another interval of $\mathcal{I}$, and we let $\mathcal{I}^{\texttt{exposed}}(G)$ denote the set of all exposed interval of G. By the association between vertices and intervals, we will use the terms interval and vertex interchangeably. We note that by definition, the subgraph of G consisting of exposed vertices forms a proper interval graph. Let $x, y \in \mathcal{I}^{\texttt{exposed}}(G)$ be two exposed vertices, and by symmetry suppose that $l_x < l_y$. As x, y belong to a proper interval graph, we also have $r_x < r_y$. We will use $x < y$ to denote that $l_x < l_y$ for two exposed vertices, and $<$ defines a strict total order on the exposed vertices of the interval graph G (i.e. $<$ simply makes a comparison on the left endpoints of the given vertices, which must be unique). The following lemma allows us to reduce the distance query on interval graphs to the proper interval graph on the exposed vertices only.

▶ **Lemma 14.** *Given an interval graph G with fixed interval representation $\mathcal{I}$, its exposed intervals $\mathcal{I}^{\texttt{exposed}}(G)$ form a proper interval graph H with the following properties:*

- *Any interval x is contained by an interval of G iff x is contained by an interval of H.*
- *For any two vertices $x, y \in G$, $\texttt{dist}_G(x, y) = \texttt{dist}_{H \cup \{x,y\}}(x, y)$.*

Intuitively, we only need exposed intervals because, by definition of `parent` (equation 1), all `parent` intervals are exposed. Thus, Lemma 14 implies that, to support `dist` on interval graphs, it suffices to maintain an instance of fully dynamic proper interval graphs using Theorem 11, on the exposed vertices of G. Thus it remains to determine and maintain exposed vertices. To do so, in addition to using an instance of the fully dynamic proper interval graph structure, we will also store the exposed intervals in an auxiliary data structure:

▶ **Lemma 15.** *There exists a data structure using $O(n)$ words where n is the number of intervals current in the data structure, that can store the exposed intervals and support the following operations given an interval x in $O(\lg n)$ time: **1)** determine whether x is contained by some interval currently in the data structure. **2)** report all intervals that are contained by x and delete all of them, in $O(k \lg n)$ time, where k is the number of deleted intervals. **3)** If x does not contain and is not contained by any interval currently in the data structure, insert it. **4)** If x is in the data structure, return its predecessor or successor with respect to $<$, or report that it doesn't exist.*

We can use a red-black tree to store the exposed intervals using the left end points as the keys and the right end points as the values. This is sufficient to support the operations in Lemma 15. We now sketch our support for dynamic interval graphs under only insertions.

Proof Sketch of Theorem 12. We maintain the fully dynamic proper interval graph structure of Theorem 11 on the graph H whose vertices are the exposed intervals of G. We also maintain the binary search tree, T_H in Lemma 15 on the same intervals. Upon the insertion of an interval, if it is contained in some other interval, then we do nothing. If the new interval is exposed, then any interval it contains will no longer be exposed and must be deleted. As the deletion of intervals from the proper interval graph structure uses $O(\lg n)$ time and any exposed interval can only be deleted once, the total time over n insertions is $O(n \lg n)$. ◀

4.1 Data Structure for Decremental Interval Graphs

In this subsection, we consider the decremental case, where the updates are the deletion of intervals. First, we investigate how the set of exposed intervals change after a deletion.

▶ **Lemma 16.** *Let G be a proper interval graph, and $G' = G - x$ for some vertex $x \in G$. If x is not exposed in G, then, $\mathcal{I}^{\texttt{exposed}}(G) = \mathcal{I}^{\texttt{exposed}}(G')$.*

If x is exposed in G, then $\mathcal{I}^{\texttt{exposed}}(G) \setminus \{x\} \subseteq \mathcal{I}^{\texttt{exposed}}(G')$, and, for all $y \in \mathcal{I}^{\texttt{exposed}}(G') \setminus \mathcal{I}^{\texttt{exposed}}(G)$, $y \subseteq x$. Furthermore, $\mathcal{I}^{\texttt{exposed}}(G') = (\mathcal{I}^{\texttt{exposed}}(G) \setminus \{x\}) \dot\cup \{y \in \mathcal{I}^{\texttt{exposed}}(G') : x^- < y < x^+\}$ ($\dot\cup$ denotes a disjoint union), where x^- and x^+ are respectively the predecessor and the successor of x in $\mathcal{I}^{\texttt{exposed}}(G)$ (if either x^- or x^+ does not exist, then that constraint is omitted). That is, the newly added elements of $\mathcal{I}^{\texttt{exposed}}(G')$ are between x^- and x^+.

In plain words, the lemma states that any new exposed intervals must fall between the predecessor and successor of the removed exposed interval.

We will now assume that the deleted vertex x is exposed, as there is nothing to be done if not. We wish to find set $\{y \in \mathcal{I}^{\texttt{exposed}}(G') : x^- < y < x^+\}$. To do so, we will iteratively find these newly exposed intervals from the smallest to the largest.

▶ **Lemma 17.** *Consider a set of intervals S, and let $x \in S$ be an exposed interval. Let $x' \in S$ be the interval with minimum $l_{x'}$ (ties broken by largest $r_{x'}$) such that $r_{x'} > r_x$. Then, x' is exposed, and there does not exist any exposed interval $z \in S$ such that $x < z < x'$. If no such x' exists, then there is no exposed interval $w \in S$ such that $x < w$.*

Next we give a data structure which applies the criteria given in Lemma 17.

▶ **Lemma 18.** *There is an $O(n)$-word data structure that can maintain a set of intervals (initially a given set, and not necessarily exposed) and support the following operations*

1. *Delete an interval in $O(\lg n)$ time;*
2. *Given any two exposed intervals x and y where $x < y$, report all exposed intervals z such that $x < z < y$ in $O((k+1)\lg n)$ time, where k is the number of returned intervals. We also allow $x = -\infty$ and/or $y = \infty$, in which case the constraint involving them is omitted.*

The structure is an augmented red-black tree storing all intervals, where the keys are the right endpoints of intervals (ties are broken by largest left end points of intervals), and to support the second operation, at each node, we store the minimum left endpoint of all the intervals in the subtree. Finally, we give a sketch of the data structure for supporting `delete`.

Proof sketch of Theorem 13. We build the data structures for the incremental case and also maintain T_G, the binary search tree in Lemma 18 on all the intervals of G. If the vertex x to be deleted is not exposed, then nothing needs to be done. If it is, then we find its predecessor and successor in $\mathcal{I}^{\texttt{exposed}}(G)$ using T_H, delete it, and find all the newly exposed vertices using Lemma 17 and T_G, and add them into the proper interval graph. As every interval can become exposed at most once, it is added into the proper interval graph at most once (and deleted at most once). Hence, over n `deletes`, the total run time is $O(n \lg n)$. ◀

5 Fully Dynamic Interval Graphs

The main goal of this section is to prove the following result:

▶ **Theorem 19.** *An interval graph G can be represented in $O(n)$ words to support* `dist` *in $O(n \lg n/S(n))$ time,* `shortest_path` *in $O(\lg n)$ time per vertex on the path, and* `insert` *and* `delete` *in $O(S(n)+\lg n)$ time, where n is the number of vertices currently in G and $S(n)$ is an arbitrary function that satisfies $S(n) = \Omega(1)$ and $S(n) = O(n)$. Setting $S(n) = \sqrt{n \lg n}$ yields an $O(n)$-word solution with $O(\sqrt{n \lg n})$-time support for* `dist`*,* `insert` *and* `delete`*.*

To prove this result, we first design a data structure to compute `parent` under updates. We decompose distance queries so that the structures can be more easily updated in Sections 5.1 and 5.2. Finally, we show how to update these in Section 5.3.

We define the `jump` of i as moving from the current vertex i to its `parent`(i) (as in Lemma 3) [3]. Then we can interpret Lemma 3 to a series of `jumps` from u to u_i, then to v. First, we give a data structure computing `jump`. By definition of `parent` (equation 1), the parent of a vertex i is the vertex $j = \arg\min_{r_j \geq l_i} l_j$. To compute this, we store all intervals in a red-black tree with their right endpoints as keys. At each node, we store both the minimal left endpoint among all intervals stored in the subtree rooted at that node and a pointer to the node containing that interval. We call this the *global interval tree*, and it uses $O(n)$ words. Given an interval i, to find j, we search for l_i to obtain a path (i.e. the nodes encountered on the standard binary search algorithm) and a set of subtrees (i.e. the subtrees rooted at right children of any node on the path, given that path continues towards the left child) containing the intervals v with $r_v \geq l_i$. For each we calculate the maximal left endpoint contained in the node (for those on the path) or the subtree and take the minimum. As the tree is balanced, this takes $O(\lg n)$ time. This tree can also be updated upon interval insertion or deletion in $O(\lg n)$ time. This simple structure is also a dynamic shortest path oracle for interval graphs.

5.1 Distance Computation

In this section, we will give an algorithm to compute the distance between two vertices x and y that is compatible with updates. The main idea is to break the interval graph into blocks of size $S(n)$. If we take the naive approach of calculating a shortest path, then the time complexity to compute the distance between two vertices x and y will depend on the distance itself. To combat this, we will decompose the path into subpaths each residing entirely within one of the blocks, and compute them quickly.

We sort the n given intervals by left endpoints, fix $S(n)$ and then divide the vertices (consecutively) into $\Theta(\frac{n}{S(n)})$ blocks, with each block containing $O(S(n))$ vertices (intervals). From left to right, we number blocks incrementally by $B_1, B_2, \ldots$. For vertex i we say that the `jump` $i \to$ `parent`(i) is an *in-block jump* if both i and `parent`(i) belong to the same block, and it is an *out-of-block jump* otherwise.

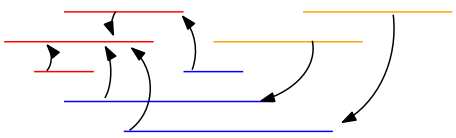


Figure 2 Depicting the parent relationship where the intervals are coloured to depict the decomposition into blocks of size 3.

[3] Though `jump`$(i) =$ `parent`(i), it flows more naturally when we describe it as an action, as it is a verb.

▶ **Example 20.** Consider figure 2. We see that any jump from the vertices represented by the red intervals are in-block jump, while jumps from all other vertices are out-of-block jumps. In particular, for any other vertex, since the jump is out-of-block, their parent in the distance tree of their block T_i (to be constructed in Section 5.2) would be different than their parent over all. However, for the red vertices these two parent relationships would coincide.

For any shortest path $x = p_1, \dots p_k, y$ where $p_{i+1} = \texttt{parent}(p_i)$, we can decompose it into a sequence of in-block jumps followed by an out-of-block jump - and repeat. To compute the path length, we will compute the length of each of the in-block sequences and count the out-of-block jumps using

- `compressed-in-block-jump`(x, optional: y): Return the last vertex t such that all jumps between $x, \dots, t$ are all in-block jumps, together with the distance between x and t. If y is given, also returns the distance between x and y or report that there is no path between them in the block.
- `jump`(x): given a vertex x, return its (global) `parent`.

Thus given two vertices x and y (with $l_y < l_x$ so that the block of x comes no earlier than the block of y) to the distance problem, we propose algorithm 1.

Algorithm 1 Compressed computation of the distance between vertices x and y.

```
1:  p, dist ← x, 0
2:  while p and y are not in the same block do
3:      q, d(p,q) ← compressed-in-block-jump(p)
4:      if q is adjacent to y then
5:          return dist + d(p,q) + 1
6:      p, dist ← parent(q), dist + d(p,q) + 1
7:      if p = q (i.e. q = parent(q)) then
8:          return unreachable
9:      if p is adjacent to y then
10:         return dist + 1
11: q, d(p,q), d(p,y) ← compressed-in-block-jump(p,y)
12: if l_y < l_q then
13:     if parent(q) is adjacent to y then
14:         return dist + d(p,q) + 1
15:     else
16:         return unreachable
17: return dist + d(p,y)
```

To show that this algorithm is correct, first we note that the blocks we visit is (weakly) monotonic. That is, at each jump, the block number never increases. Hence after each cycle of `compressed-in-block-jump` and `jump` the block number always decreases. This can be seen as by definition, for any x, `parent`(x) has a smaller left endpoint, and as our blocks are obtained by sorting the left endpoints, it cannot be in a larger numbered block.

Proof. (Correctness of Algorithm 1) Let $s_1 > s_2 > s_3 \dots$ be the sequence of block numbers in our jump sequence. Let p_i be the first vertex in our path of block s_i and $q_i =$ `compressed-in-block-jump`(p_i) be the last vertex of block s_i. Let t be the block number of y. We have two cases: If $s_i > t > s_{i+1}$ for some i, then either q_i is adjacent to y or p_{i+1} is necessarily adjacent to y, the first we check on line 4, the second on line 9. The edge case here is if $s_i > t$ for all i, then at the end q_i would have no parent (which we defined

as $\texttt{parent}(q_i) = q_i$) as it is the end of its component, at which point we return that it is unreachable (on line 7-8). If $t = s_i$ for some i, then once we hit p_i, the while loop ends and we compute the remaining distance within the block. If q_i, y are in the same block with $l_{q_i} \leq l_y$, then the distance computation between p_i and y use only in-block `jump`s and thus is correct (line 17). Otherwise, if $l_{q_i} > l_y$, then we have the same situation as above. If it has a parent, then the parent must be adjacent to y (line 13-14). If it does not, then y cannot be reached (line 16). ◀

Given this, the time complexity of the distance algorithm is upper bounded by the time complexity of `compressed-in-block-jump` and `parent` multiplied by $O(\frac{n}{S(n)})$, as each operation is called at most $O(\frac{n}{S(n)})$ (the number of blocks) times.

5.2 Data Structures for Analyzing Jumps in Block

Now, we will discuss how to implement the `compressed-in-block-jump` subroutine after some preprocessing of each block. For each block B_i, we construct the distance tree, T_i (which may be a forest) of the interval graph G_i induced by the intervals in B_i. We precompute the depth, $\texttt{depth}(T_i, x)$, of each node x in T_i and also preprocess T_i using the approach of Bender and Farach-Colton [6] to provide constant-time support for the level ancestor operator, $\texttt{anc}(T_i, x, d)$, which, given a node x in T_i, returns the ancestor of x at depth d of T_i. The preprocessing time is $O(|B_i|)$, and T_i uses $O(|B_i|)$ words of space after preprocessing. By Lemma 3, `depth` and `anc` are sufficient to compute in constant time the distance, $\texttt{dist}(G_i, x, y)$, of two vertices x and y in G_i or determine that there is no path between them (i.e. when the two vertices belong to different trees in the forest). We will also store all the intervals of B_i using the left endpoint as keys in a red-black tree.

We propose the following `compressed-in-block-jump` algorithm (Algorithm 2).

Algorithm 2 `compressed-in-block-jump`(x, optional: y).

1: **if** y exists **then**
2: **return** $\texttt{dist}(G_i, x, y)$ and also perform the below steps
3: Compute $L_{limit}(i)$
4: Compute $v_{limit}(i)$
5: **if** $L_{limit}(i) \geq x$ **then**
6: **return** $x, 0$ as x jumps out of block already
7: Apply Lemma 3 to obtain z (u_i in Lemma 3) the last node before $v_{limit}(i)$ on the path between x and $v_{limit}(i)$ - or unreachable if a path cannot be found (i.e. they are in different subtrees).
8: **if** Line 7 is unreachable **then**
9: **return** $\texttt{anc}(T_i, x, 0), \texttt{depth}(x)$
10: **if** $l_z \leq L_{limit}$ **then**
11: **return** $z, \texttt{depth}_{T_i}(x) - \texttt{depth}_{T_i}(z)$
12: **else**
13: **return** $\texttt{parent}_{T_i}(z), \texttt{depth}_{T_i}(x) - \texttt{depth}_{T_i}(z) + 1$

Define $L_{limit}(i) = \max\{r_w \mid l_w < \min\{l_v \mid v \in B_i\}\}$ and, if $L_{limit}(i) < \min\{l_v \mid v \in B_i\}$, we set it as $-\infty$ for convenience. This has the property that, for any $v \in B_i$ with $l_v < L_{limit}(i)$, $\texttt{jump}(v)$ is out-of-block as w (the interval achieving the maximum value in the definition of $L_{limit}(i)$) is a candidate for $\texttt{parent}(v)$ and is out-of-block. Conversely, $\texttt{jump}(v)$ for any

v with $l_v > L_{limit}(i)$ is in-block since any out of block jump from v would move $L_{limit}(i)$ to its right. Let $v_{limit}(i) = \arg\max_{\{v \in B_i | l_v < L_{limit}(i)\}} l_v$ be the last out-of-block `jump` vertex, computed by searching for $L_{limit}(i)$ over the left endpoints of the intervals in the block.

We may compute $L_{limit}(i)$ by descending the global interval tree. First search for the value $\min\{l_v \mid v \in B_i\}$ to obtain a path and a set of subtrees as right children of nodes on the path - these are all the intervals with right endpoint greater than $\min\{l_v \mid v \in B_i\}$. For each subtree and node on the path (in reverse in-order: starting with the subtree/node containing the interval with the largest right endpoint), we check if the minimal left endpoint in the subtree (or the left endpoint of the interval for a node on the path) is less than $\min\{l_v \mid v \in B_i\}$. If not, then we move on to the next subtree or node on the path. If so we return this interval for a node, and for a subtree we traverse it: for any node, if the right children's minimal left endpoint is less than $\min\{l_v \mid v \in B_i\}$, we move to it. Otherwise, we check the current node's interval and return it if its left endpoint is less than $\min\{l_v \mid v \in B_i\}$. If not, we move to the left child. If at the end, none of the subtrees nor nodes have a left endpoint smaller than $\min\{l_v \mid v \in B_i\}$, we return $-\infty$.

To find the first vertex with an out-of-block jump we compute $L_{limit}(i)$ and $v_{limit}(i)$. Then we apply $\texttt{dist}(v_{limit}(i), x)$ as in Lemma 3. Since $v_{limit}(i)$ is the boundary between the in-block jumps and out-of-block jumps, The penultimate vertex on the path to $v_{limit}(i)$ (i.e. u_i in Lemma 3) will also be on the boundary between being in-block and out-of-block (i.e. either it is the first out-of-block jump or it is the last in-block jump, and we check which by its relative order with $v_{limit}(i)$).

Correctness of `compressed-in-block-jump`**.** For computing the distance between x and y in the block, we call $\texttt{dist}(G_i, x, y)$ directly, which has been analyzed in Lemma 3 as a constant time operation with the constant-time support for `depth` and `anc` in T_i.

Furthermore, we want the number of `jump`s before we reach past $v_{limit}(i)$. If x and $v_{limit}(i)$ are in different trees in the forest T_i, then either x jumps out of block already, or we cannot reach $v_{limit}(i)$ via in-block jumps starting from x and the sequence of jumps ends at the root of the tree containing x. If they are in the same tree, we compute the path from x to $v_{limit}(i)$ and find the penultimate node on the path x_d (i.e. u_i in Lemma 3). If x_d jumps out of block, then x_{d-1} cannot, since otherwise it would also be adjacent to $v_{limit}(i)$. Similarly, if x_d does not jump out of block, then, as $\texttt{parent}(x_d) \leq v_{limit}(i)$ ($v_{limit}(i)$ is a candidate for the parent of x_d), $\texttt{parent}(x_d)$ must jump out of block. ◀

The query time of `compressed-in-block-jump` is $O(\lg n)$ since, all steps taken, such as the computation of $L_{limit}(i)$ and `parent`, are $O(\lg n)$ time. As the time complexity of the distance algorithm is the time complexity of `compressed-in-block-jump` and `parent` multiplied by $O(\frac{n}{S(n)})$, Algorithm 1 uses $O(n \lg n / S(n))$ time to answer a distance query.

5.3 Maintaining Data Structure under Update Operations

Update operations include vertex (interval) insertions and deletions. For any update, we have to maintain the following structures:

1. The global data structure of `parent` computation (i.e. the global interval tree).
2. The local `parent` structure of each block (i.e T_i for each block B_i).
3. The block structure of the whole graph.

As discussed the global interval tree can be maintained in $O(\lg n)$ time for each update. This part is independent of the block structure. As the structure for each local block only uses the intervals of that block, for each update, we need only update the structures for the

block containing the inserted or deleted interval. Since the update will change the interval set of this block, we rebuild the local structures in $O(S(n))$ time. Thus overall each update has complexity $O(S(n) + \lg n)$.

Note that we assume the block size stays $O(S(n))$ for the previous analysis. However, the block size will change whenever an update occurs in it. In the worst case, all updates would occur in the same block, and thus we must be able to maintain our block sizes to be $\Theta(S(n))$. Moreover, the total number of intervals n will also change, and thus $S(n)$ will also change. The efficiency may not be guaranteed if the total number of intervals no longer correlates with our block size. Therefore, we use two processes, `Split` and `Rebuild`, to maintain block sizes of $\Theta(S(n))$. In general, `Split` happens whenever a block's size is too large, which might degenerate the complexity of `compressed-in-block-jump`. A `Rebuild` will occur after a certain number of updates in order to control the number of blocks and ensure that the block size corresponds to the current $S(n)$ value.

Split: The key part of the analysis is that we assume the block size is $O(S(n))$ all the time. The `Split` process is for maintaining this property. After any `insert`, if the size of the block containing the new vertex is $2S(n)$, we will split the block into two blocks, and each block has $S(n)$ intervals sorted by left endpoints. As we need to rebuild two blocks of size $S(n)$, the time needed is $O(S(n))$. Finally, we note that since every block begins with $S(n)$ vertices and has $S(n)$ vertices after a split, at least $S(n)$ inserts must occur in a block to split it.

Rebuild: This process denotes a complete rebuilding of the whole block structures, including block dividing and all local preprocessing of blocks. Since the cost of building the structure for each block is linear, the total cost is $O(n)$. The motivation of this operation comes from two causes. Firstly, when the number of blocks increases greatly, the compressed distance computing, which is dominated by the number of blocks, will degenerate. Secondly, denoting the number of intervals in the last `Rebuild` as n', if n' differs greatly from n, $S(n)$ may also be different enough from $S(n')$, so that our complexity will not be $O(n \lg n/S(n))$ per query or $O(S(n) + \lg n)$ per update (corresponding to the current $S(n)$). Therefore, we trigger a `Rebuild` after every $\frac{n'}{2}$ updates. After all $\frac{n'}{2}$ updates, the current number of intervals n stays within $[\frac{n'}{2}, \frac{3n'}{2}]$, and at most $\frac{n'}{2S(n')}$ blocks are created or destroyed. To see this, to destroy a block requires $S(n)$ deletions and to add a block also needs $S(n)$ insertions to trigger a `Split`. Thus the number of blocks remains $\Theta(\frac{n'}{S(n')}) = \Theta(\frac{n}{S(n)})$. Since $S(n)$ is a function that satisfies $S(n) = O(n)$, the complexity of all these n' updates and queries in between stays the same corresponding to the previous $S(n')$.

To deamortize, whenever we `Rebuild`, we create a new structure over the next $\frac{n}{4} = \Theta(n')$ updates containing the contents of the original structure (using the new $S(n)$ as our block size) and the $\frac{n}{4}$ updates. While we rebuild, we also perform the updates in the old structure and answer queries using the old structure. Thus for each of the next $\frac{n}{4}$ updates, we incur an extra $O(S(n'))$ time per update. Upon the completion of the rebuild, we switch to using the newly created structure. Lastly, we summarize every part and prove the main theorem here.

Proof of Theorem 19. For an interval graph G that the number of vertices is currently n, in our algorithm, we only maintain a global structure to maintain `parent`s and do local preprocessing for answering `depth` and `anc` in each block. The `parent` data structure takes $O(n)$ space. The preprocessing of each of the $\Theta(\frac{n}{S(n)})$ blocks occupies $O(T)$ words (T denotes the number of intervals in this block). The aggregation gives $O(n)$ space in total. Since $S(n) = O(n)$, the interval graph is therefore represented in $O(n)$ words of space. Distance

queries takes $O(n \lg n/S(n))$ time. For any update, we showed in Section 5.3, that it takes $O(S(n) + \lg n)$ time. Moreover, the extra `Split` and `Rebuild` processes only guarantee the cost is not degenerated and does not affect the complexity per operation. Hence, we have our proof for the main theorem. ◀

6 A Lower Bound for Axis-Aligned Line Segments in 3D

In this section, we show that the problem of supporting distance queries over dynamic intersection graphs of 3D axis-aligned line segments is conditionally hard by reducing from the online matrix-vector multiplication problem.

▶ **Definition 21** (Online Boolean Matrix Vector Multiplication (OMv)). *Let M be a $n \times n$ boolean matrix, and let $v_1, \ldots, v_n$ be a set of $n \times 1$ vectors. We must compute and output Mv_i for each i before receiving the next vector.*

The corresponding hardness conjecture is by Henzinger et al. [19]. For any constant $\varepsilon > 0$, there is no $O(n^{3-\varepsilon})$-time algorithm that solves OMv with error probability at most 1/3.

▶ **Theorem 22.** *If updates and distance queries over an intersection graph of 3D axis-aligned line segments can be respectively supported in $O(Q(n))$ and $O(T(n))$ time, then OMv can be solved in $O(n^2(T(n) + Q(n)))$ time. Thus, for any constant $\varepsilon > 0$, $T(n)$ and $Q(n)$ can not both be $O(n^{1-\varepsilon})$, unless the OMv conjecture is false. Here n is the length of the vectors. If $\hat{n}$ is the number of vertices of the graph, then $T(\hat{n})$ and $Q(\hat{n})$ cannot both be $O(\hat{n}^{1/2-\varepsilon})$.*

Proof. For each $i \in [1, n]$, define a line segment X_i between end points $(0, i, 0)$ and $(2n, i, 0)$, and a segment Y_i between $(i, 0, 1)$ and $(i, 2n, 1)$. For each entry $M_{i,j} = 1$, we create a line segment $Z_{j,i}$ between $(j, i, 0)$ and $(j, i, 1)$. We will refer to the 3 types of line segments as type X, type Y and type Z segments. Furthermore, we add the line segment X_{n+1} whose end points are $(0, n + 1, 0)$ and $(2n, n + 1, 0)$ to represent the incoming vector. When given a vector v_i, we add the following type Z segments. For each entry $v_i(j) = 1$, we add a type Z segment $Z_{j,n+1}$ with end points $(j, n + 1, 0)$ and $(j, n + 1, 1)$. By construction, both type X and type Y segments are only adjacent to type Z segments, and each type Z segment $Z_{i,j}$ is only adjacent to two other segments X_j and Y_i.

We now claim that $Mv_i(j) = 1$ if and only if $\texttt{dist}(X_j, X_{n+1}) = 4$. First suppose that $\texttt{dist}(X_j, X_{n+1}) = 4$. Then by construction, there exists an index w such that the path is $X_j, Z_{w,j}, Y_w, Z_{w,n+1}, X_{n+1}$. This implies that $M(j, w) = 1$ and $v_i(w) = 1$, and thus $Mv_i(j) = 1$. Conversely, suppose that $Mv_i(j) = 1$. Then there exists an index w such that $M(j, w) = v_i(w) = 1$. Thus $X_j, Z_{w,j}, Y_w, Z_{w,n+1}, X_{n+1}$ is a path of length 4, so $\texttt{dist}(X_j, X_{n+1}) \leq 4$. To see that it cannot be strictly less than 4, we note that, since each type Z segment is adjacent to a single type X segment and a single type Y segment, we may view it as a subdivision of an edge between its two incident segments. Since segments of types X (and Y) are never adjacent to segments of the same type, if we contract vertices representing type Z segments, we are left with a bipartite graph. If $\texttt{dist}_G(X_j, X_{n+1}) < 4$, it must be 2 (by the subdivision of edges), but it cannot be 2 as that implies two type X segments are adjacent.

Thus, the computation of a single Mv_i operation is reduced to $O(n)$ insertions and deletions of segments of type Z, and n distance queries. Over all n operations, this incurs $O(n^2)$ updates and $O(n^2)$ queries. Therefore, $O(n^2T(n) + n^2Q(n))$ cannot be $O(n^{3-\varepsilon})$ for any constant $\varepsilon > 0$ unless the OMv conjecture is false. Finally, we note that by construction, the number of vertices in the graph is at most $\hat{n} = O(n^2)$ and the theorem follows. ◀

References

1 Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In David Peleg, editor, *Distributed Computing – 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings*, volume 6950 of *Lecture Notes in Computer Science*, pages 404–415. Springer, 2011. doi:10.1007/978-3-642-24100-0_39.

2 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.

3 Josh Alman, Timothy Chu, Aaron Schild, and Zhao Song. Algorithms and hardness for linear algebra on geometric graphs. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 541–552. IEEE, 2020. doi:10.1109/FOCS46700.2020.00057.

4 Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Maintaining information in fully-dynamic trees with top trees. *ACM Transactions on Algorithms*, 1, December 2003. doi:10.1145/1103963.1103966.

5 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.

6 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.

7 Karl Bringmann, Sándor Kisfaludi-Bak, Marvin Künnemann, André Nusser, and Zahra Parsaeian. Towards sub-quadratic diameter computation in geometric intersection graphs. In Xavier Goaoc and Michael Kerber, editors, *38th International Symposium on Computational Geometry, SoCG 2022, June 7-10, 2022, Berlin, Germany*, volume 224 of *LIPIcs*, pages 21:1–21:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SoCG.2022.21.

8 Timothy M. Chan. Finding triangles and other small subgraphs in geometric intersection graphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1777–1805. SIAM, 2023. doi:10.1137/1.9781611977554.ch68.

9 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in geometric intersection graphs. *J. Comput. Geom.*, 10(1):27–41, 2019. doi:10.20382/jocg.v10i1a2.

10 Timothy M. Chan and Dimitrios Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. *J. Comput. Geom.*, 10(2):3–20, 2019. doi:10.20382/jocg.v10i2a2.

11 Danny Z. Chen, D. T. Lee, R. Sridhar, and Chandra N. Sekharan. Solving the all-pair shortest path query problem on interval and circular-arc graphs. *Networks*, 31(4):249–258, 1998. doi:10.1002/(SICI)1097-0037(199807)31:4<249::AID-NET5>3.0.CO;2-D.

12 Jonathan B. Conroy and Csaba D. Tóth. Hop-spanners for geometric intersection graphs. In Xavier Goaoc and Michael Kerber, editors, *38th International Symposium on Computational Geometry, SoCG 2022, June 7-10, 2022, Berlin, Germany*, volume 224 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SoCG.2022.30.

13 Christophe Crespelle. Fully dynamic representations of interval graphs. *Theoretical Computer Science*, 759:14–49, 2019. doi:10.1016/j.tcs.2019.01.007.

14 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004. doi:10.1145/1039488.1039492.

15 Hicham El-Zein, Moshe Lewenstein, J Ian Munro, Venkatesh Raman, and Timothy M Chan. On the succinct representation of equivalence classes. *Algorithmica*, 78:1020–1040, 2017.

16 Jie Gao and Li Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM J. Comput.*, 35(1):151–169, 2005. doi:10.1137/S0097539703436357.

17 Cyril Gavoille and Christophe Paul. Optimal distance labeling for interval graphs and related graph families. *SIAM J. Discret. Math.*, 22(3):1239–1258, 2008. doi:10.1137/050635006.

18 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.

19 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 the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, STOC '15, pages 21–30, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746609.

20 Hung Le and Christian Wulff-Nilsen. Optimal approximate distance oracle for planar graphs. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 363–374. IEEE, 2021. doi:10.1109/FOCS52979.2021.00044.

21 Yaowei Long and Seth Pettie. Planar distance oracles with better time-space tradeoffs. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 2517–2537. SIAM, 2021. doi:10.1137/1.9781611976465.149.

22 J. Ian Munro and Corwin Sinnamon. Time and space efficient representations of distributive lattices. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 550–567. SIAM, 2018. doi:10.1137/1.9781611975031.36.

23 J. Ian Munro and Kaiyu Wu. Succinct data structures for chordal graphs. 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 67:1–67:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

24 Mihai Pătraşcu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. *SIAM J. Comput.*, 43(1):300–311, 2014. doi:10.1137/11084128X.

25 Gaurav Singh, N. S. Narayanaswamy, and G. Ramakrishna. Approximate distance oracle in $o(n^2)$ time and $o(n)$ space for chordal graphs. In M. Sohel Rahman and Etsuji Tomita, editors, *WALCOM: Algorithms and Computation – 9th International Workshop, WALCOM 2015, Dhaka, Bangladesh, February 26-28, 2015. Proceedings*, volume 8973 of *Lecture Notes in Computer Science*, pages 89–100. Springer, 2015. doi:10.1007/978-3-319-15612-5_9.

26 Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In Torben Hagerup and Jyrki Katajainen, editors, *Algorithm Theory – SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Humlebaek, Denmark, July 8-10, 2004, Proceedings*, volume 3111 of *Lecture Notes in Computer Science*, pages 384–396. Springer, 2004. doi:10.1007/978-3-540-27810-8_33.

27 Antti Ukkonen, Carlos Castillo, Debora Donato, and Aristides Gionis. Searching the wikipedia with contextual information. In James G. Shanahan, Sihem Amer-Yahia, Ioana Manolescu, Yi Zhang, David A. Evans, Aleksander Kolcz, Key-Sun Choi, and Abdur Chowdhury, editors, *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008*, pages 1351–1352. ACM, 2008. doi:10.1145/1458082.1458274.

28 Monique V. Vieira, Bruno M. Fonseca, Rodrigo Damazio, Paulo Braz Golgher, Davi de Castro Reis, and Berthier A. Ribeiro-Neto. Efficient search ranking in social networks. In Mário J. Silva, Alberto H. F. Laender, Ricardo A. Baeza-Yates, Deborah L. McGuinness, Bjørn Olstad, Øystein Haug Olsen, and André O. Falcão, editors, *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, CIKM 2007, Lisbon, Portugal, November 6-10, 2007*, pages 563–572. ACM, 2007. doi:10.1145/1321440.1321520.

29 Harry Wiener. Structural determination of paraffin boiling points. *J. Am. Chem. Soc.*, 69(1):17–20, 1947.

30 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.

FPT Approximation Using Treewidth: Capacitated Vertex Cover, Target Set Selection and Vector Dominating Set

Huairui Chu
Nanjing University, China

Bingkai Lin
Nanjing University, China

Abstract

Treewidth is a useful tool in designing graph algorithms. Although many NP-hard graph problems can be solved in linear time when the input graphs have small treewidth, there are problems which remain hard on graphs of bounded treewidth. In this paper, we consider three vertex selection problems that are W[1]-hard when parameterized by the treewidth of the input graph, namely the capacitated vertex cover problem, the target set selection problem and the vector dominating set problem. We provide two new methods to obtain FPT approximation algorithms for these problems. For the capacitated vertex cover problem and the vector dominating set problem, we obtain $(1 + o(1))$-approximation FPT algorithms. For the target set selection problem, we give an FPT algorithm providing a tradeoff between its running time and the approximation ratio.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases FPT approximation algorithm, Treewidth, Capacitated vertex cover, Target set selection, Vector dominating set

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.19

1 Introduction

We consider problems whose goals are to select a minimum sized vertex set in the input graph that can "cover" all the target objects. In the capacitated vertex cover problem (CVC), we are given a graph G with a capacity function $c : V(G) \to \mathbb{N}$, the goal is to find a set $S \subseteq V(G)$ of minimum size such that every edge of G is covered[1] by some vertex in S and each vertex $v \in S$ covers at most $c(v)$ edges. This problem has application in planning experiments on redesign of known drugs involving glycoproteins [24]. In the target set selection problem (TSS), we are given a graph G with a threshold function $t : V(G) \to \mathbb{N}$. The goal is to select a minimum sized set $S \subseteq V(G)$ of vertices that can activate all the vertices of G. The activation process is defined as follows. Initially, all vertices in the selected set S are activated. In each round, a vertex v gets active if there are $t(v)$ activated vertices in its neighbors. The study of TSS has application in maximizing influence in social network [26]. Vector dominating set (VDS) can be regarded as a "one-round-spread" version of TSS, where the input consists of a graph G and a threshold function $t : V(G) \to \mathbb{N}$, and the goal is to find a set $S \subseteq V(G)$ such that for all vertices $v \in V$, there are at least $t(v)$ neighbors of v in S.

Since CVC generalizes the vertex cover problem, while TSS and VDS are no easier than the dominating set problem[2], they are both NP-hard and thus have no polynomial time algorithm unless $P = NP$. Polynomial time approximation algorithms for capacitated vertex

[1] An edge e can be covered by a vertex v if v is an endpoint of e.
[2] For VDS, when $t(v) = 1$ for every vertex v in the graph, VDS is the dominating set problem. For TSS, a reduction from dominating set to TSS can be found in the work of Charikar et al. [8].

cover problem have been studied extensively [24, 12, 22, 34, 11, 36, 35]. The problem has a 2-approximation polynomial time algorithm [22]. Assuming the *Unique Game Conjecture*, there is no polynomial time algorithm for the vertex cover problem with approximation ratio better than 2 [27]. As for the TSS problem, it is known that the minimization version of TSS cannot be approximated to $2^{\log^{1-\epsilon} n}$ assuming $P \neq NP$, or $n^{0.5-\epsilon}$ assuming the conjecture on planted dense subgraph [10, 9]. Cicalese et al. proved that VDS cannot be approximated within a factor of $c \ln n$ for some c unless $P = NP$ [13].

Another way of dealing with hard computational problems is to use parameterized algorithms. For any input instance x with parameter k, an algorithm with running time upper bounded by $f(k) \cdot |x|^{O(1)}$ for some computable function f is called FPT. A natural parameter for a computational problem is the solution size. The first FPT algorithm with running time $1.2^{k^2} + n^2$ for capacitated vertex cover problem parameterized by solution size was provided in [25]. In [17], the authors gave an improved FPT algorithm that runs in $k^{3k} \cdot |G|^{O(1)}$. However, using the solution size as parameter might be too strict for CVC. Note that CVC instances with sublinear capacity functions cannot have small sized solutions. On the other hand, TSS parameterized by its solution size is W[P]-hard [3] according to [1]. VDS is W[2]-hard since it generalizes the dominating set problem.

In this paper, we consider these problems parameterized by the treewidth [33] of the input graph. In fact, since the treewidth of a graph having k-sized vertex cover is also upper-bounded by k [17], CVC parameterized by treewidth can be regarded as a natural generalization of CVC parameterized by solution size. And it is already proved in [17] that CVC parameterized only by the treewidth of its input graph has no FPT algorithm assuming $W[1] \neq FPT$. As for the TSS problem, it can be solved in $n^{O(w)}$ time for graphs with n vertices and treewidth bounded by w and has no $n^{o(\sqrt{w})}$-time algorithm unless ETH fails [3]. VDS is also $W[1]$-hard when parameterized by treewidth [4], however, it admits an FPT algorithm with respect to the combined parameter $(w + k)$[32].

Recently, the approach of combining parameterized algorithms and approximation algorithms has received increased attention [19]. It is natural to ask whether there exist FPT algorithms for these problems with approximation ratios better than that of the polynomial time algorithms. Lampis [29] proposed a general framework for approximation algorithms on tree decomposition. Using his framework, one can obtain algorithms for CVC and VDS which outputs a solution of size at most $opt(I)$ on input instance I but may slightly violate the capacity or the threshold requirement within a factor of $(1 \pm \epsilon)$. However, the framework of Lampis can not be directly used to find an approximation solution for these problems satisfying all the capacity or threshold requirement. The situation becomes worse in the TSS problem, as the error might propagate during the activation process. We overcome these difficulties and give positive answer to the aforementioned question. For the CVC and VDS problems, we obtain $(1 + o(1))$-approximation FPT algorithms respectively.

▶ **Theorem 1.** *There exists an algorithm* [4]*, which takes a CVC instance $I = (G, c)$ and a tree decomposition $(T, \mathcal{X})$ with width w for G as input and outputs an integer $\hat{k}_{\min} \in [opt(I), (1 + O(1/(w^2 \log n)))opt(I)]$ in $(w \log n)^{O(w)} n^{O(1)}$ time.*

▶ **Theorem 2.** *There exists an algorithm running in time $2^{O(w^5 \log w \log\log n)} n^{O(1)}$ which takes as input an instance $I = (G, t)$ of VDS and a tree decomposition of G with width w, finds a solution of size at most $(1 + 1/(w \log\log n)^{\Omega(1)}) \cdot opt(I)$.*

---

[3] The well known W-hierarchy is $FPT \subseteq W[1] \subseteq W[2] \subseteq ... \subseteq W[P]$, where FPT denotes the set of problems which admits FPT algorithms. The basic conjecture on parameterized complexity is $FPT \neq W[1]$. We refer the readers to [18, 20, 15] for more details.

[4] The algorithm can be modified to output a solution with size as promised. See the remark in Appendix B.

Notice that the running time stated in above theorems are FPT running time, because $(\log n)^{f(w)} \leq f(w)^{O(f(w))} + n^{O(1)}$.

For the TSS problem, we give an approximation algorithm with a tradeoff between the approximation ratio and its running time.

▶ **Theorem 3.** *For all $C \in \mathbb{N}$, there is an algorithm which takes as input an instance $I = (G, t)$ of TSS and a tree decomposition of G with width w, finds a solution of size $(1 + (w+1)/(C+1)) \cdot opt(I)$ in time $n^{C+O(1)}$.*

Open problems and future work. Note that our FPT approximation algorithm for TSS has ratio equal to the treewidth of the input graph. An immediate question is whether this problem has parameterized $(1 + o(1))$-approximation algorithm. We remark that the reduction from k-Clique to TSS in [3] does not preserve the gap. Thus it does not rule out constant FPT approximation algorithm for TSS on bounded treewidth graphs even under hypotheses such as *parameterized inapproximablity hypothesis* (PIH) [30] or GAP-ETH [16, 31].

In the regime of exact algorithms, we have the famous Courcelle's Theorem which states that all problems defined in *monadic second order logic* have linear time algorithm on graphs of bounded treewidth [2, 14]. It is interesting to ask if one can obtain a similar algorithmic meta-theorem [23] for approximation algorithms.

1.1 Overview of our techiniques

Capacitated Vertex Cover. Our starting point is the exact algorithm for CVC on graphs with treewidth w in $n^{\Theta(w)}$ time. The exact algorithm has running time $n^{\Theta(w)}$ because it has to maintain a set of $(w+1)$-dimension vectors $d : X_\alpha \to [n]$ for every node α in the tree decomposition. One can get more insight by checking out the exact algorithm for CVC in Section 3. To reduce the size of such a table, Lampis' approach [29] is to pick a parameter $\epsilon \in (0, 1)$ and round every integer to the closest integer power of $(1 + \epsilon)$. In other words, an integer n is represented by $(1 + \epsilon)^x$ with $(1 + \epsilon)^x \leq n < (1 + \epsilon)^{x+1}$. Thus it suffices to keep $(\log n)^{O(w)}$ records for every bag in the tree decomposition. The price of this approach is that we can only have approximate values for records in the table. Note that the errors of approximate values might accumulate after addition (See Lemma 9). Nevertheless, we can choose a tree decomposition with height $O(w^2 \log n)$ and set $\epsilon = 1/poly(w \log n)$ so that if the dynamic programming procedure only involves adding and passing values of these vectors, then we can have $(1 + o(1))$-approximation values for all the records in the table.

Unfortunately, in the forgetting node for a vertex v, we need to compare the value of $d(v)$ and the capacity value $c(v)$. This task seems impossible if we do not have the exact value of $d(v)$. Our idea is to modify the "slightly-violating-capacity" solution, based on two crucial observations. The first is that, in a solution, for any vertex $v \in V$, the number of edges incident to v which are **not** covered by v presents a lower bound for the solution size. The second observation is that one can test if a "slightly-violating-capacity" solution can be turned into a good one in polynomial time. These observations are formally presented in Lemma 10 and 11.

Target Set Selection and Vector Dominating Set. We observe that both of the TSS and VDS problems are *monotone* and *splittable*, where the monotone property states that any super set of a solution is still a solution and the splittable property means that for any separator X of the input graph G, the union of X and solutions for components after removing X is also a solution for the graph G. We give a general approximation for vertex

subset problems that are monotone and splittable. The key of our approximation algorithm is an observation that any bag in a tree decomposition is a separator in G. As the problem is splittable, we can design a procedure to find a bag, and remove it, which leads to a separation of G and we then deal with the component "rooted" by this bag. We can use this procedure repeatedly until the whole graph is done.

1.2 Organization of the Paper

In Section 2 the basic notations are given, and we formally define the problem we study. In Section 3 we present the exact algorithm for CVC. In Section 4 we present the approximate algorithm for CVC. In Section 5, we give the approximation algorithms for TSS and VDS.

2 Preliminaries

2.1 Basic Notations

We denote an undirected simple graph by $G = (V, E)$, where $V = [n]$ for some $n \in \mathbb{N}$ and $E \subseteq \binom{V}{2}$. Let $V(G) = V$ and $E(G) = E$ be its vertex set and edge set. For any vertex subset $S \subseteq V$, let the induced subgraph of S be $G[S]$. The edges of $G[S]$ are $E[S] = E(G) \cap \binom{S}{2}$. For any $S_1, S_2 \subseteq V$, we use $E[S_1, S_2]$ to denote the edge set between S_1 and S_2, i.e. $E[S_1, S_2] = \{e = (u, v) \in E \mid u \in S_1, v \in S_2\}$. For every $v \in V(G)$, we use $N(v)$ to denote the neighbors of v, and $d(v) := |N(v)|$.

For an orientation O of a graph G, which can be regarded as a directed graph whose underlying undirected graph is G, we use $D^+_O(v)$ to denote the outdegree of v and $D^-_O(v)$ its indegree. In a directed graph or an orientation, an edge (u, v) is said to start at u and sink at v. Reversing an edge is an operation, in which an edge (u, v) is replaced by (v, u).

In a graph $G = (V, E)$, a separator is a vertex set X such that $G[V \setminus X]$ is not a connected graph. In this case we say X separates V into disconnected parts $C_1, C_2, ... \subseteq V \setminus X$, where C_i and C_j are disconnected for all $i \neq j$ in $G[V \setminus X]$.

Let $f : A \to B$ be a mapping. For a subset $A' \subseteq A$, let $f[A']$ denote the mapping with domain A' and $f[A'](a) = f(a)$, for all $a \in A'$. Let $f \setminus a$ be $f[A \setminus \{a\}]$. For all $b \in B$, let $f^{-1}(b)$ be the set $\{a \in A' \mid f(a) = b\}$.

Let $\gamma \geq 0$ be a small value, we use $\mathbb{N}_\gamma$ to denote $\{0\} \cup \{(1 + \gamma)^x \mid x \in \mathbb{N}\}$. For $a, b \in \mathbb{R}$, we use $a \sim_\gamma b$ to denote that $b/(1 + \gamma) \leq a \leq (1 + \gamma)b$. It's easy to see this is a symmetric relation. Further, we use $[a]_\gamma$ to denote $\max_{x \in \mathbb{N}_\gamma, x \leq a} x$. Notice that $[a]_\gamma \sim_\gamma a$.

2.2 Problems

Capacitated Vertex Cover. An instance of CVC consists of a graph $G = (V, E)$ and a capacity function $c : V \to \mathbb{N}$ on the vertices. A solution is a pair (S, M) where $S \subseteq V$ and $M : E \to S$ is a mapping. If for all $v \in S, |M^{-1}(v)| \leq c(v)$ and for all $e \in E, M(e) \in e$, then we say that S is feasible. The size of a feasible solution is $|S|$. The goal of CVC is to find a feasible solution of minimum size. An equivalent description of this problem is the following. Let O be an orientation of all the edges in E. O is a feasible solution if and only if for all $v \in V, D^-_O(v) \leq c(v)$. The size of O is defined as $|\{v \in V \mid d^-(v) > 0\}|$. Here we actually use a directed edge (u, v) to represent that $\{u, v\}$ is covered by v. We mainly use this definition as it's more convenient for organizing our proof and analysis.

Target Set Selection. Given a graph $G = (V, E)$, a threshold function $t : V \to \mathbb{N}$, and a set $S \subseteq V$, the set $S' \subseteq V$ which contains the vertices activated by S is the smallest set that:

- $S \subseteq S'$;
- For a vertex v, if $|N(V) \cap S'| \geq t(v)$, then $v \in S'$.

One can find the vertices activated by S in polynomial time. Just start from $S' := S$, as long as there exists a vertex v such that $|N(v) \cap S'| \geq t(v)$, add v to S', until no such vertex exists. A vertex set that can activate all vertices in V is called a target set. The goal of TSS is to find a target set of minimum size.

Vector Dominating Set. Given a graph $G = (V, E)$, a threshold function $t : V \to \mathbb{N}$, the goal of Vector Dominating Set problem is to find a minimum vertex subset $S \subseteq V$ such that every vertex $v \in V \setminus S$ satisfies $|N(v) \cap S| \geq t(v)$.

2.3 Tree Decomposition

In this paper, we consider problems parameterized by the treewidth of the input graphs. A tree decomposition of a graph G is a pair $(T, \mathcal{X})$ such that

- T is a rooted tree and $\mathcal{X} = \{X_\alpha : \alpha \in V(T), X_\alpha \subseteq V(G)\}$ is a collection of subsets of $V(G)$;
- $\bigcup_{\alpha \in V(T)} X_\alpha = V(G)$;
- For every edge e of G, there exists an $\alpha \in V(T)$ such that $e \subseteq X_\alpha$;
- For every vertex v of G, the set $\{\alpha \in V(T) \mid v \in X_\alpha\}$ forms a subtree of T.

The width of a tree decomposition $(T, \mathcal{X})$ is $\max_{\alpha \in V(T)} |X_\alpha| - 1$. The treewidth of a graph G is the minimum width over all its tree decompositions.

The sets in $\mathcal{X}$ are called "bags". For a node $\alpha \in V(T)$, let T_α denote the subtree of T rooted by α. Let $V_\alpha \subseteq V$ denote the vertex set $V_\alpha = \cup_{\alpha' \in V(T_\alpha)} X_{\alpha'}$. Let $Y_\alpha := V_\alpha \setminus X_\alpha$. For a node α, we use $\alpha_1(, \alpha_2)$ to denote its possible children. By the definition of tree decompositions, for a join node α, $Y_{\alpha_1} \cap Y_{\alpha_2} = \emptyset$.

It is convenient to work on a *nice tree decomposition*. Every node $\alpha \in V(T)$ in this nice tree decomposition is expected to be one of the following:

(i) Leaf Node: α is a leaf and $X_\alpha = \emptyset$;

(ii) Introducing v Node: α has exactly one child α_1, $v \notin X_{\alpha_1}$ and $X_\alpha = X_{\alpha_1} \cup \{v\}$;

(iii) Forgetting v Node: α has exactly one child α_1, $v \notin X_\alpha$ and $X_\alpha \cup \{v\} = X_{\alpha_1}$;

(iv) Join Node: α has exactly two children α_1, α_2 and $X_\alpha = X_{\alpha_1} = X_{\alpha_2}$.

Treewidth is a popular parameter to consider because tree decompositions with optimal or approximate treewidth can be efficiently computed [28]. We refer the reader to [15, 6, 5] for more details of treewidth and nice tree decomposition. Using the tree balancing technique [7] and the method of introducing new nodes, we can transform any tree decomposition with width w in polynomial time into a nice tree decomposition with width $O(w)$, depth upper bounded by $O(w^2 \log n)$, and containing at most $O(nw)$ nodes. Moreover, we can add $O(w)$ nodes so that the root α_0 is assigned with an empty set $X_{\alpha_0} = \emptyset$. Notice that in this case, $Y_{\alpha_0} = V_{\alpha_0} = V$. We assume all the nice tree decompositions discussed in this paper satisfy these properties.

3 Exact Algorithm for CVC

We present the exact algorithm for two reasons. The first is that one can gain some basic insights on the structure of the approximate algorithm by understanding the exact algorithm, which is more comprehensible. The other is that we need to compare the intermediate results of the exact algorithm and the approximate algorithm, so the total description of the algorithm can also be regarded as a recursive definition of the intermediate results (which are the sets R_α's defined in the following).

3.1 Definition of the Tables

Given a tree decomposition $(T, \mathcal{X})$, we run a classical bottom-up dynamic program to solve CVC. That is, on each node α we allocate a record set R_α. R_α contains records of the form (d, k). A record (d, k) consists of two elements: a mapping $d : X_\alpha \to \mathbb{N}$ and an integer $k \in \mathbb{N}$. At first, we present a definition of R_α by its properties. Then we define R_α according to the **Recursive Rules**. If our goal is only to design an exact algorithm for CVC, then there could be many different definitions of the tables which all work. However, here our definitions are elaborated so that they fit in our analysis of the approximation algorithm. After these definitions are given, later in Theorem 5 we prove that these two definitions coincide.

Let G_α denote the graph with vertex set V_α and edge set $E[V_\alpha] \setminus E[X_\alpha]$. We expect that the table R_α has the following properties.

3.1.1 Expected Properties for R_α

A record $(d, k) \in R_\alpha$ if and only if there exists O, an orientation of G_α, such that
(1) For each $v \in X_\alpha$, $d(v) = D^+_O(v)$ is just its out degree;
(2) $D^-_O(v) \le c(v)$ for all $v \in Y_\alpha$;
(3) $|\{v \in Y_\alpha \mid D^-_O(v) > 0\}| \le k \le |Y_\alpha|$.
Intuitively, $(d, k) \in R_\alpha$ if there exists a vertex set $S \subseteq Y_\alpha$ and a mapping $M : E[V_\alpha] \setminus E[X_\alpha] \to S \cup X_\alpha$ such that

- all edges are covered correctly, i.e. $M(e) \in e$ for all $e \in E[V_\alpha] \setminus E[X_\alpha]$;
- for each $v \in X_\alpha$, there are $d(v)$ edges from v to Y_α that are covered by S, i.e. $|E[\{v\}, Y_\alpha] \cap \cup_{u \in S} M^{-1}(u)| = d(v)$;
- M satisfies the capacity constraints for vertices in Y_α, i.e. for all $v \in Y_\alpha$, $|M^{-1}(v)| \le c(v)$;
- $|S| \le k \le |Y_\alpha|$.

One can imagine that S is a feasible solution for a spanning subgraph of G_α, where the vector d governs the edges between X_α and Y_α.

Note that the root node α_0 satisfies $X_{\alpha_0} = \emptyset$, and $G_{\alpha_0} = G$. So if R_{α_0} is correctly computed, then the k values in those records in R_{α_0} have a one-to-one correspondence to all feasible solution sizes for the original instance. We output $\min_{(d,k) \in R_{\alpha_0}} k$ to solve the instance.

3.1.2 Recursive Rules for R_α

Fix a node $\alpha \in V(T)$, if α is a introducing node or a forgetting node, let α_1 be its child. If α is a join node, let α_1, α_2 be its children. In case α is a:

Leaf Node. $R_\alpha = \{(d, k)\}$, in which d is a mapping with empty domain and $k := 0$.

Introducing v Node. Note that by the properties of tree decompositions, there is no edge between v and Y_α in G. A record $(d, k) \in R_\alpha$ if and only if $(d \setminus v, k) \in R_{\alpha_1}$ and $d(v) = 0$.

Join Node. $(d,k) \in R_\alpha$ if and only if there exist $(d_1,k_1) \in R_{\alpha_1}$ and $(d_2,k_2) \in R_{\alpha_2}$ such that for all $v \in X_\alpha$, $d(v) = d_1(v) + d_2(v)$ and $k = k_1 + k_2$.

Forgetting v Node. $(d,k) \in R_\alpha$ if and only if there exists $(d_1,k_1) \in R_{\alpha_1}$ satisfying one of the following conditions:

(1) $k_1 = k$, $d_1(v) = |N(v) \cap Y_\alpha|$ and $d_1 \setminus v = d$. In this case, v is not "included in S". All the edges between v and Y_α must be covered by other vertices in Y_α.

(2) $k_1 = k-1$ and there exist $\Delta(v) \subseteq N(v) \cap X_\alpha$ and $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$ such that $d_1(v) = A$, $d_1(u) = d(u) - 1$ for all $u \in \Delta(v)$, and $d_1(u) = d(u)$ for all $u \in X_{\alpha_1} \setminus (\Delta(v) \cup \{v\})$. In this case, v is "included in S". We enumerate a set $\Delta(v) \subseteq N(v) \cap X_\alpha$ of edges between v and X_α and let v cover these edges. Note that for a record $(d_1,k_1) \in R_{\alpha_1}$, there are $|N(v) \cap Y_\alpha| - d_1(v)$ edges that are covered by v. To construct (d,k) from (d_1,k_1), we need to check that $c(v) \geq |\Delta(v)| + |N(v) \cap Y_\alpha| - d_1(v)$, which is implicitly done by the setting $d_1(v) = A \geq |N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|$.

▶ Remark 4. In fact, one can find many different ways to define the dynamic programming table for CVC. We use this definition because we want to upper bound the values of records in R_α by the size of solution (Lemma 10), so we need to record "outdegrees" rather than "indegrees" or "capacities".

Valid certificate. Notice that all the rules are of the form $(d_1,k_1) \in R_{\alpha_1} \Rightarrow (d,k) \in R_\alpha$ or $(d_1,k_1) \in R_{\alpha_1} \wedge (d_2,k_2) \in R_{\alpha_2} \Rightarrow (d,k) \in R_\alpha$, thus a rule can actually be divided in to two parts: we found a "valid certificate" $(d_1,k_1) \in R_{\alpha_1}$ (and $(d_2,k_2) \in R_{\alpha_2}$, for join nodes), then we add a "product" $(d,k) \in R_\alpha$ based on the certificate. In fact, every record in R_{α_1} can be a valid certificate in introducing nodes, and every pair of records $((d_1,k_1),(d_2,k_2)) \in R_{\alpha_1} \times R_{\alpha_2}$ can be a valid certificate in join nodes. But in forgetting v nodes, we further require that $d_1(v)$ satisfies some condition. To be specific, in a forgetting node α_1 we say $(d_1,k_1) \in R_{\alpha_1}$ is valid if it satisfies the following condition:

($\star$) $d_1(v) = |N(v) \cap Y_\alpha|$ or $\geq |N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|$ for some $\Delta(v) \subseteq N(v) \cap X_\alpha$.

▶ **Theorem 5.** *The set $\{R_\alpha : \alpha \in V(T)\}$ can be computed by the recursive rules above in time $n^{w+O(1)}$, and the **Expected Properties** are satisfied.*

The proof sketch of the correctness of these rules are presented in Appendix A. As $|R_\alpha| \leq n^{w+2}$ for all $\alpha \in V(T)$ and the enumerating $\Delta(v)$ procedure in dealing with a forgetting node runs in time $w^{O(w)}$, it's not hard to see that this algorithm runs in time $n^{w+O(1)}$ (for w small enough compared to n).

4 Approximation Algorithm for CVC

Let ϵ be a small value to be determined later. We try to compute an approximate record set $\hat{R}_\alpha$ for each node α, still using bottom-up dynamic programming like what we do in the exact algorithm. An approximate record is a pair $(\hat{d},\hat{k})$, where $\hat{k} \in \mathbb{N}$ and $\hat{d}$ is a mapping from X_α to $\mathbb{N}_\epsilon = \{0\} \cup \{(1+\epsilon)^x \mid x \in \mathbb{N}\}$. As we can see, $\hat{d}$ can take non-integer values.

Height of a Node. The height h of a node α is defined by the length of the longest path from α to a leaf which is descendent to α. By this definition, the height of a node is 1 plus the maximum height among the heights of its children. Let the height of the root node be h_0. According to the property of nice tree decompositions, h_0 is at most $O(w^2 \log n)$.

Let ϵ_h, δ_h be two non-negative values (which are functions of h, n and w) to be determined later.

***h*-close records.** If an exact record (d, k) and an approximate record $(\hat{d}, \hat{k})$ satisfy $d(v) \sim_{\epsilon_h} \hat{d}(v)$ for all $v \in X_\alpha$ and $k \sim_{\delta_h} \hat{k}$, then we say these two records are h-close.

We expect that for each node α, $\hat{R}_\alpha$ satisfies the following. Let the height of α be h.
(A) If $(d, k) \in R_\alpha$, then there exists $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$ which is h-close to (d, k).
(B) If $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$, then there exists $(d, k) \in R_\alpha$ which is h-close to $(\hat{d}, \hat{k})$.
After $\hat{R}_{\alpha_0}$ is correctly computed (i.e. satisfying (A) and (B)), we output the value $\hat{k}_{\min} = (1 + \delta_{h_0}) \min_{(\hat{d},\hat{k}) \in \hat{R}_{\alpha_0}} \hat{k}$. Let OPT be the size of the minimum solution, which equals to $\min_{(d,k) \in R_{\alpha_0}} k$. We claim that $\hat{k}_{\min} \in [OPT, (1 + \delta_{h_0})^2 OPT]$.

Proof. By property (B), we have $OPT \leq (1 + \delta_{h_0}) \min_{(\hat{d},\hat{k}) \in \hat{R}_{\alpha_0}} \hat{k}$. By property (A), we have $\min_{(\hat{d},\hat{k}) \in \hat{R}_{\alpha_0}} \hat{k} \leq (1 + \delta_{h_0}) OPT$. The claim follows by combining these two inequalities. ◀

We need the following procedure to test in polynomial time if a sub-problem is solvable when we are allowed to use all vertices to cover the edges.

▶ **Lemma 6.** *Testing whether $(d, |Y_\alpha|) \in R_\alpha$ for any d can be done in $n^{O(1)}$ time.*

Proof. Construct a directed graph with vertex set $\{s, t\} \cup (E[V_\alpha] \setminus E[X_\alpha]) \cup V_\alpha$. For each $e \in (E[V_\alpha] \setminus E[X_\alpha])$ add an edge (s, e) with capacity 1. For each $e = (u, v) \in (E[Y_\alpha] \setminus E[X_\alpha])$ add an edge (e, u) and an edge (e, v) both with capacity 1. For each $v \in X_\alpha$ add an edge (v, t) with capacity $|N(v) \cap Y_\alpha| - d(v)$. For each $v \in Y_\alpha$ add an edge (v, t) with capacity $c(v)$. We claim that $(d, |Y_\alpha|) \in R_\alpha$ if and only if there is a flow from s to t with value $|E[V_\alpha] \setminus E[X_\alpha]|$. For the 'if' part, notice that by the well-known integrality theorem for network flow, there exists a integral flow with the same value. Every integral flow with this value can be transform to an O as expected in the **Expected Properties**: An edge $e \in E[Y_\alpha] \setminus E[X_\alpha]$ is oriented so that it sinks at vertex v if (e, v) has flow value 1, then for each vertex $v \in X_\alpha$, reverse some edges in $E[\{v\}, Y_\alpha]$ so that $D_O^+(v) = d(v)$, if the flow carried in (v, t) is less than $|N(v) \cap Y_\alpha| - d(v)$. One can construct a flow with the value based on an orientation O, too. Thus the 'only if' part is easy to see, too. ◀

We first define $\{\hat{R}_\alpha : \alpha \in V(T)\}$ using the following **Recursive Rules**. Then we prove that these sets satisfy the properties (A) and (B). The basic idea of our approximate algorithm is to run the exact algorithm in an "approximate way". For a rule formed as $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1} \Rightarrow (\hat{d}, \hat{k}) \in \hat{R}_\alpha$ or $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1} \wedge (\hat{d}_2, \hat{k}_2) \in \hat{R}_{\alpha_2} \Rightarrow (\hat{d}, \hat{k}) \in \hat{R}_\alpha$, we also call $(\hat{d}_1, \hat{k}_1)$ (and $(\hat{d}_2, \hat{k}_2)$) the certificate while $(\hat{d}, \hat{k})$ is the product.

4.1 Recursive Rules for $\hat{R}_\alpha$

Fix a node $\alpha \in V(T)$ with height h, in case α is a:

Leaf Node. $\hat{R}_\alpha = \{(\hat{d}, \hat{k})\}$, in which $\hat{d}$ is a mapping with empty domain and $\hat{k} = 0$.

Introducing v Node. A record $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$ if and only if $(\hat{d} \setminus v, \hat{k}) \in \hat{R}_{\alpha_1}$ and $\hat{d}(v) = 0$.

Join Node. $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$ if and only if there exists $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}, (\hat{d}_2, \hat{k}_2) \in \hat{R}_{\alpha_2}$ such that for each $v \in X_\alpha$, $\hat{d}(v) = [\hat{d}_1(v) + \hat{d}_2(v)]_\epsilon$ and $\hat{k} = \hat{k}_1 + \hat{k}_2$.

Forgetting v Node. This case is the most complicated. Think in this way: we pick $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$ and based on it we try to construct $(\hat{d}, \hat{k})$ to add into $\hat{R}_\alpha$. Notice that in the exact algorithm, not every $(d_1, k_1) \in R_{\alpha_1}$ can be used to generate a corresponding product $(d, k) \in R_\alpha$ – it has to be the case that $d_1(v) = |N(v) \cap Y_\alpha|$ or $d_1(v) \geq |N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|$, which is what we called to be a valid certificate. We have to test both the validity of the certificate and its exact counterpart using an indirect way. So there are three issues we need to address:

(a) The requirement for $(\hat{d}_1, \hat{k}_1)$ being valid, i.e. satisfying the "approximate version" of condition $(\star)$;

(b) There exists a valid exact counterpart $(d_1, k_1) \in R_{\alpha_1}$ of $(\hat{d}_1, \hat{k}_1)$ satisfying condition $(\star)$;

(c) How to construct $(\hat{d}, \hat{k})$.

(b) seems impossible since we do not compute R_{α_1}, we obtain this indirectly using Lemma 6. Later we explain why such an approach reaches our goal. Formally, suppose we have $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$, we consider two cases:

(1) v is not "included".

(1a) See if $\hat{d}_1(v) \sim_{\epsilon_{h-1}} |N(v) \cap Y_\alpha|$;

(1b) See if $(d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, where $d_t(u) = \lceil \hat{d}_1(u)/(1+\epsilon_{h-1}) \rceil$ for all $u \in X_{\alpha_1} \setminus \{v\}$ and $d_t(v) = |N(v) \cap Y_\alpha|$ (This is polynomial-time tractable by Lemma 6);

(1c) If (a) and (b) are satisfied, then add $(\hat{d}, \hat{k})$ to $\hat{R}_\alpha$, where $\hat{d} = \hat{d}_1 \setminus v, \hat{k} = \hat{k}_1$.

(2) v is "included". We enumerate $\Delta(v) \subseteq N(v) \cap X_\alpha$ and integer A satisfying $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$.

(2a) See if $\hat{d}_1(v) \geq A/(1+\epsilon_{h-1})$;

(2b) See if $(d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, where $d_t(u) = \lceil \hat{d}_1(u)/(1+\epsilon_{h-1}) \rceil$ for all $u \in X_{\alpha_1} \setminus \{v\}$ and $d_t(v) = A$ (By Lemma 6, this is still polynomial-time tractable);

(2c) If (a) and (b) are satisfied, then add $(\hat{d}, \hat{k})$ to $\hat{R}_\alpha$, where $\hat{d}(u) = \hat{d}_1(u)$ for all $u \in X_\alpha \setminus \Delta(v)$, $\hat{d}(u) = [\hat{d}(u)+1]_\epsilon$ for all $u \in \Delta(v)$, $\hat{k} = \hat{k}_1 + 1$.

▶ **Theorem 7.** *Set $\epsilon = \frac{1}{(w^2 \log n)^3}$, $\epsilon_h = 2h\epsilon$ and $\delta_h = 4(h+1)h\epsilon$. Suppose n is great enough. When the dynamic programming is done, for all α, $\hat{R}_\alpha$ satisfies property (A) and (B).*

Proof of Theorem 1. According to Theorem 7 and the above discussion, we immediately get $\hat{k}_{\min} \in [OPT, (1+\delta_{h_0})^2 OPT]$. By the property of nice tree decomposition, h_0 is at most $O(w^2 \log n)$, thus $\hat{k}_{\min} \in [OPT, (1+O(1/(w^2 \log n)))^2 OPT] = [OPT, (1+O(1/(w^2 \log n)))OPT]$.

The space we need to memorize each $\hat{R}_\alpha$ is $O((w^6 \log^4 n)^w n^{O(1)})$. Computing a leaf/introduce/join node we need $O((w^6 \log^4 n)^{2w} n^{O(1)})$ time. In a forgetting node, we may need to enumerate some set $\Delta(v) \subseteq N(v) \cap X_\alpha$, which requires time $O(2^{|X_\alpha|}) = O(2^{w+1})$. So computing a Forgetting node requires $O((w^6 \log^4 n)^w 2^w n^{O(1)})$ time. The tree size is polynomial, so the total running time is FPT. ◀

To prove Theorem 7, we need a few lemmas. The proof of Lemma 8 and Lemma 9 are presented in Appendix B. Lemma 8 and Lemma 9 are some simple observations. To understand why we need Lemma 10 and Lemma 11, remember that we have a complicated recursive rule for forgetting nodes in which we verifies (a) and (b). However, we cannot directly verify if a valid exact record described in (b) exists, because we don't have R_{α_1}. We overcome this by verifies a feasible partial solution (e.g. $(d_1, |Y_{\alpha_1}|)$ in (1b)) rather than an optimal one, which can be done by Lemma 6. When we are computing $\hat{R}_\alpha$, we assume that $\hat{R}_{\alpha_1}$ has been correctly computed, i.e. it satisfies (A) and (B). So there exists $(d_1, k_1) \in rcds1$ which is $h-1$-close to $(\hat{d}_1, \hat{k}_1)$. However, we don't know if (d_1, k_1) is a so-called valid certificate. Lemma 11 shows how to modify (d_1, k_1) so that it becomes we want in (b), knowing $(d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$. We introduces some error like $o(1)d_1(v)$ on k_1 in this procedure. Lemma 10 helps us to rewrite it as $o(1)k_1$.

▶ **Lemma 8.** *If $(d, k) \in R_\alpha$ for some node α, then for every (d', k') with $d(v) \geq d'(v)$ for all $v \in X_\alpha$ and k' satisfying $k \leq k' \leq |Y_\alpha|$, we have $(d', k') \in R_\alpha$.*

▶ **Lemma 9.** *Let $a, b, a', b' \in \mathbb{R}, h \in \mathbb{N}^+$, $\epsilon_h \in [0,1]$, $a' \sim_{\epsilon_h} a$ and $b' \sim_{\epsilon_h} b$. Then we have $[a'+b']_\epsilon \sim_{\epsilon_{h+1}} (a+b)$.*

▶ **Lemma 10.** *For all $(d,k) \in R_\alpha$ and $v \in X_\alpha, k \geq d(v)$.*

Proof. Let O be the orientation. Let $N^+(v) = \{u \in V(G) : (v,u) \in E(G)\}$ be out neighbors of v. By definition, we have $d(v) = |N^+(v)| \leq |\{u \in Y_\alpha \mid D^-_O(u) > 0\}| \leq k$. ◀

▶ **Lemma 11.** *Fix some $(d,k) \in R_\alpha, v \in X_\alpha$ and some integer $p > 0$ satisfying $k + p \leq |Y_\alpha|$. Let $d_m : X_\alpha \to \mathbb{N}$ be a function such that $d_m(v) = d(v) + p$ and $d_m \setminus v = d \setminus v$. We have $(d_m, |Y_\alpha|) \in R_\alpha$ if and only if $(d_m, k+p) \in R_\alpha$.*

Proof. On one hand, the 'if' part is obvious by Lemma 8. On the other hand, we prove that $(d_m, |Y_\alpha|) \in R_\alpha$ implies $(d', k+1) \in R_\alpha$, where $d'(v) = d(v) + 1, d' \setminus v = d \setminus v$. Then we can repeatedly increase the value of k by 1 for p times to obtain the 'only if' part. Let the orientation corresponding to (d,k) and $(d_m, |Y_\alpha|)$ be O_1, O_2 respectively. Now let G' be a graph with vertex set $Y_\alpha \cup \{v\}$. A directed edge (x,y) is in G' if and only if $(x,y) \in O_2$ and $(y,x) \in O_1$.

By picking O_1 so that the number of edges in G' is minimized, we can assume that G' contains no cycle. Otherwise if G' contains a cycle, we can reverse every edge along the cycle in O_1 so that it is still a valid orientation for (d,k) but the number of edges in G' decreases.

As $D^+_{O_2}(v) > D^+_{O_1}(v)$, there exists an non-empty path in G' starting from v ending at, say, $v' \neq v$ such that v' has no out edge in G'. This implies $D^-_{O_1}(v') \leq D^-_{O_2}(v') - 1$, or v' will have an out edge in G'. We reverse the edges along this path in O_1. Let the new orientation be O_3. $D^-_{O_3}(v') \leq D^-_{O_1}(v') + 1 \leq D^-_{O_2}(v') \leq c(v)$. Moreover, $\{u \mid D^-_{O_3}(u) > 0\} \setminus \{u \mid D^-_{O_1}(u) > 0\} \subseteq \{v'\}$. Thus, O_3 is a valid orientation for $(d', k+1)$. ◀

4.2 Theorem 7 Proof Sketch

Due to space limit, the complete proof is presented in Appendix B.2.

We use induction on nodes, following a bottom-up order on the tree decomposition. Leaf nodes satisfy property (A) and (B), because $R_\alpha = \hat{R}_\alpha$ for every leaf node. Fix a node α of height h, by induction, we assume that every node descendent to α satisfies (A) and (B). We only need to prove that α satisfies both (A) and (B). We make a case distinction based on the type of α. The case where α is a forgetting node is the most complicated and requires lemma 10 and 11. The other two types follow Lampis' framework.

To show α satisfies (A), we need to prove the existence of some $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$ for any given $(d,k) \in R_\alpha$ such that $(\hat{d}, \hat{k})$ and (d,k) are h-close. This is done by first picking up the certificate of (d,k), that is, the record $(d_1, k_1) \in R_{\alpha_1}$ (or a pair of records in the case α is a join node, we omit join node case in the following sketch) which "produces" (d,k) based on recursive rules for R_α. Then by induction hypothesis, there is an $(h-1)$-close record $(\hat{d}_1, \hat{k}_1)$ in $\hat{R}_{\alpha_1}$. If α is not a forgetting node, then according to recursive rules for $\hat{R}_\alpha$, there exists $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$. We prove that $(\hat{d}, \hat{k})$ and (d,k) are h-close. If α is a forgetting node, then we verify (1b) or (2b) by applying Lemma 8 on (d_1, k_1).

To show α satisfies (B), if α is not a forgetting node, then we pick up and compare some records in a different order: We start from $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$; Then we pick $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$ according to recursive rules for $\hat{R}_\alpha$; Next we pick $(d_1, k_1) \in R_{\alpha_1}$ based on induction hypothesis; Finally we find out $(d,k) \in R_\alpha$ using recursive rules for R_α. If α is a forgetting node, suppose the record $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$ is produced by $(\hat{d}_1, \hat{k}_1)$. The main idea is to apply Lemma 11 on $(d_t, |Y_{\alpha_1}|)$, the record verified by (1b) or (2b), and (d_1, k_1), the record $(h-1)$-close to $(\hat{d}_1, \hat{k}_1)$, so as to show the existence of some $(d,k) \in R_\alpha$. At the same time we use Lemma 10 to bound k.

5 Approximation algorithms for TSS and VDS

In this section, we introduce the *vertex subset problem* which is a generalization of many graph problems. Then we present a sufficient condition for the existence of parameterized approximation algorithms for such problems parameterized by the treewidth. Finally, we apply our algorithm to target set selection problem (TSS) and vector dominating set problem (VDS), which are both vertex subset problems satisfying this condition. The definitions bellow are inspired by Fomin, et al. [21].

▶ **Definition 12** (Vertex Subset Problem). *A vertex subset problem Φ takes a string $I \in \{0,1\}^*$ as an input, which encodes a graph $G_I = (V_I, E_I)$ and some possible additional information, e.g. threshold values on vertices. Φ is identified by a function $\mathcal{F}_\Phi$ which maps a string $I \in \{0,1\}^*$ to a family of vertex subsets of V_I, say $\mathcal{F}_\Phi(I) \subseteq 2^{V_I}$. Any vertex set in $\mathcal{F}_\Phi(I)$ is a **solution** of the instance I. The goal is to find a minimum sized solution.*

We will often select a set of vertices and assume that it is included in a solution, and then consider the remaining sub-problem. So we introduce the concept of partial instances.

▶ **Definition 13** (Partial Vertex Subset Problem). *Let Φ be a vertex subset problem. The partial version of Φ takes a string $I \in \{0,1\}^*$ appended with a vertex subset $U \subseteq V_I$ as input. We call such a pair (I, U) a partial instance of Φ. Any vertex set $W \subseteq V_I \setminus U$ is a solution if and only if $W \cup U \in \mathcal{F}_\Phi(I)$. Still, the goal is to find a minimum sized solution.*

We consider the following conditions of a vertex subset problem Φ.

- Φ is **monotone**, if for any instance I, $S \in \mathcal{F}_\Phi(I)$ implies for all S' satisfying $S \subseteq S' \subseteq V_I$, $S' \in \mathcal{F}_\Phi(I)$.
- Φ is **splittable**, if: for any instance I and any separator X of G_I which separates $V_I \setminus X$ into disconnected parts $V_1, V_2, ..., V_p$, if $S_1, S_2, ..., S_p$ are vertex sets such that S_i is a solution for the partial instance $(I, V_I \setminus V_i), \forall 1 \leq i \leq p$, then $X \cup \bigcup_{1 \leq i \leq p} S_i$ is a solution for I.

It is trivial to show the monotonicity for TSS and VDS. To see that they are splittable, observe that given an instance $I = (G, t)$ of VDS for example, fix some $X \subseteq V(G)$, a set S containing X is a solution for I if and only if $S \setminus X$ is a solution for $I' = (G', t')$, where $G' = G[V \setminus X]$ and $t'(v) = t(v) - |N(v) \cap X|$ for all $v \in V \setminus X$. If X is a separator, then the graph G' is not connected, and the union of any solutions of each component in G', with X together forms a solution of I. This observation also works for TSS.

The main theorem in this section is to show the tractability, in the sense of parameterized approximation, of monotone and splittable vertex subset problems on graphs with bounded treewidth.

▶ **Theorem 14.** *Let Φ be a vertex selection problem which is monotone and splittable. If there exists an algorithm such that on input a partial instance of Φ appended with a corresponding nice tree decomposition with width w, it can run in time $f(\ell, w, n)$ and*

- *either output the optimal solution, if the size of it is at most ℓ;*
- *or confirm that the optimal solution size is at least $\ell + 1$*

then there exists an approximate algorithm for Φ with ratio $1 + (w+1)/(l+1)$ and runs in time $f(l, w, n) \cdot n^{O(1)}$, for all $l \in \mathbb{N}$.

We provide a trivial algorithm for the partial version of TSS. Given a partial instance $(I = (G, t), U)$, we search for a solution of size at most ℓ by brute-force. This takes time $f(\ell, w, n) = n^{\ell + O(1)}$. Setting $l := C$ in Theorem 14, we simply get the following.

▶ **Corollary 15** (Restated version of Theorem 3). *For all constant C, TSS admits a $1+(w+1)/(C+1)$-approximation algorithm running in time $n^{C+O(1)}$.*

As mentioned before, Raman et al.[32] showed that VDS is $W[1]$-hard parameterized by w, but FPT with respect to the combined parameter $(k+w)$ where k is the solution size. The running time of their algorithm is $k^{O(wk^2)}n^{O(1)}$. A partial instance (I,U) of VDS can be transformed to an equivalent VDS instance, in which the input graph is $G[V_I \setminus U_I]$, so this algorithm can also be used for the partial version of VDS. Set $l := w^2(\log\log n/\log\log\log n)^{0.5}$ in Theorem 14, we get Corollary 16.

▶ **Corollary 16** (Restated version of Theorem 2). *Vector Dominating Set admits a $1+1/(w\log\log n)^{\Omega(1)}$-approximation algorithm running in time $2^{O(w^5\log w\log\log n)}n^{O(1)}$.*

Notice that we can't obtain a $(1+o(1))$-approximation for TSS using a similar approach, because solving TSS in $f(w+k)n^{O(1)}$-time is $W[1]$-hard [3].

One may also think of applying Theorem 14 to CVC, since CVC is FPT when parameterized by solution size [17]. However, CVC is not splittable. Think of a simple 3-vertex graph with vertex set $\{a,b,c\}$ and edge set $\{\{a,b\},\{b,c\}\}$. The capacities are: $c(a)=0, c(b)=1, c(c)=0$. $\{b\}$ is a separator in this graph and empty sets are two solutions for the two partial instances. However, $\{b\}$ cannot cover both two edges in the original graph.

5.1 The Algorithm Framework

To prove Theorem 14, we introduce the concept of l-good node.

▶ **Definition 17** (l-good Node). *Let I be an instance of a vertex selection problem Φ and $(T,\mathcal{X})$ be a nice tree decomposition of any subgraph of G_I. A node $\alpha \in V(T)$ is an l-good node if the partial instance $(I, V_I \setminus Y_\alpha)$ admits a solution of size at most l.*

For a node α, let N_α^- denote the set of all children of α. We present the pseudocode of our algorithm in Algorithm 1. Figure 1 in Appendix C illustrates how the sets defined in Algorithm 1 are related. Algorithm 1 solves the partial version of Φ. For the original version, when we get an instance I, we just create an equivalent partial instance $(I,\emptyset)$ appended with a nice tree decomposition $(T,\mathcal{X})$ and an integer l, then we run $Solve((I,\emptyset),(T,\mathcal{X}),l)$. The analyze of Algorithm 1 is presented in Appendix C.

Main idea of Algorithm 1: Let Alg be an algorithm solving partial instances in time $f(l,w,n)$. Given a partial instance (I,D) and a nice tree decomposition $(T,\mathcal{X})$ on $G[I\setminus D]$, we run Alg to test the goodness of each node. If the root node is l-good, then (I,D) has a solution with size at most l, we use Alg to find the optimal solution. If a leaf node is not l-good then by monotonicity I has no solution[5].Otherwise, we can pick a lowest node α which is not l-good. Then all its children are l-good. Such a node has nice properties.

- On one hand, since all children of α are l-good, the partial instances $(I, V_I \setminus Y_{\alpha_c})$ can be optimally solved by Alg for each α_c a child of α. Adding X_{α_c} and the optimal solution E_{α_c} for $(I, V_I \setminus Y_{\alpha_c})$ into the solution enables us to "discard" the whole subtree rooted by α_c and the corresponding vertices, i.e. V_{α_c};

[5] By our definition of vertex subset problem, the set of solutions can be empty. However any instance of TSS or VDS admits at least one solution which is the whole vertex set.

- On the other hand, as α is not l-good, by the splittable and monotone properties, we can deduce that the optimal solution S^* has an intersection of size at least $(l+1)$ with Y_α i.e. $|S^* \cap Y_\alpha| \geq l+1$.

Based on these properties, the algorithm iteratively finds one such node α and includes $X_{\alpha_c} \cup E_{\alpha_c}$ for its every child α_c into the solution, then "removes" V_{α_c} from the graph. Once we repeat this procedure, the optimal solution size decreases by at least $|S^* \cap (\bigcup_{\alpha_c} V_{\alpha_c})| \geq |S^* \cap Y_\alpha| \geq l+1$. For each α_c, we use Alg to find the optimal solution E_{α_c}, so in each Y_{α_c} we select at most $|S^* \cap Y_{\alpha_c}|$ vertices. The "non-optimal" part is $\bigcup_{\alpha_c} X_{\alpha_c}$, which is at most $O(w) = O(w/l)|S^* \cap (\bigcup_{\alpha_c} V_{\alpha_c})|$. Therefore, the approximation ratio is upper bounded by $1 + \frac{|\bigcup_{\alpha_c} X_{\alpha_c}|}{l+1} \leq 1 + O(w/l)$.

Algorithm 1 Subprocess $Solve()$.

Input: A partial instance (I, D) of Φ, a nice tree decomposition $(T, \mathcal{X})$ of $G_I[V_I \setminus D]$ with width w, $l \in \mathbb{N}$ an integer.

Output: A solution S to (I, D), or 'there exists no solution'.

for *each node* α **do**
  Use Alg to test if α is an l-good node;
  if α *is* l*-good* **then**
    $E_\alpha :=$ the minimum solution for $(I, V_I \setminus Y_\alpha)$;
  end
end
if *the root* α_0 *is* l*-good* **then**
  Return E_{α_0};
end
Find a node α which is not l-good with minimum height;
if α *is a leaf node* **then**
  Return 'there exists no solution';
end
$E' := \emptyset$;
$F := \emptyset$;
for *each* $\alpha_c \in N^-_\alpha$ **do**
  $E' := E' \cup E_{\alpha_c} \cup X_{\alpha_c}$;
  $F := F \cup V_{\alpha_c}$;
end
Find a nice tree decomposition $(T', \mathcal{X}')$ for $G_I[V_I \setminus (D \cup F)]$;
Return $E' \cup Solve((I, D \cup F), (T', \mathcal{X}'), l)$;

References

1 Karl R. Abrahamson, Rodney G. Downey, and Michael R. Fellows. Fixed-parameter tractability and completeness IV: on completeness for W[P] and PSPACE analogues. *Ann. Pure Appl. Log.*, 73(3):235–276, 1995. doi:10.1016/0168-0072(94)00034-Z.

2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

3 Oren Ben-Zwi, Danny Hermelin, Daniel Lokshtanov, and Ilan Newman. Treewidth governs the complexity of target set selection. *Discrete Optimization*, 8(1):87–96, 2011.

4 Nadja Betzler, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discrete Applied Mathematics*, 160(1-2):53–60, 2012.

5 Hans L Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 226–234, 1993.

6 Hans L Bodlaender. A tourist guide through treewidth. *Acta cybernetica*, 11(1-2):1, 1994.

7 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.

8 Moses Charikar, Yonatan Naamad, and Anthony Wirth. On approximating target set selection. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, volume 60 of *LIPIcs*, pages 4:1–4:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.APPROX-RANDOM.2016.4.

9 Moses Charikar, Yonatan Naamad, and Anthony Wirth. On approximating target set selection. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*, 2016.

10 Ning Chen. On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics*, 23(3):1400–1415, 2009.

11 Wang Chi Cheung, Michel X Goemans, and Sam Chiu-wai Wong. Improved algorithms for vertex cover with hard capacities on multigraphs and hypergraphs. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1714–1726. SIAM, 2014.

12 Julia Chuzhoy and Joseph Naor. Covering problems with hard capacities. *SIAM Journal on Computing*, 36(2):498–515, 2006.

13 Ferdinando Cicalese, Martin Milanič, and Ugo Vaccaro. On the approximability and exact algorithms for vector domination and related problems in graphs. *Discrete Applied Mathematics*, 161(6):750–767, 2013.

14 Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Formal Models and Semantics*, pages 193–242. Elsevier, 1990.

15 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

16 Irit Dinur. Mildly exponential reduction from gap 3sat to polynomial-gap label-cover. *Electron. Colloquium Comput. Complex.*, page 128, 2016. URL: https://eccc.weizmann.ac.il/report/2016/128.

17 Michael Dom, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Capacitated domination and covering: A parameterized perspective. In *International Workshop on Parameterized and Exact Computation*, pages 78–90. Springer, 2008.

18 Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

19 Andreas Emil Feldmann, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.

20 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

21 Fedor V Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. *Journal of the ACM (JACM)*, 66(2):1–23, 2019.

22 Rajiv Gandhi, Eran Halperin, Samir Khuller, Guy Kortsarz, and Aravind Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *Journal of Computer and System Sciences*, 72(1):16–33, 2006.

23 Martin Grohe. Logic, graphs, and algorithms. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 357–422. Amsterdam University Press, 2008.

24 Sudipto Guha, Refael Hassin, Samir Khuller, and Einat Or. Capacitated vertex covering with applications. In *Symposium on Discrete Algorithms: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, volume 6, pages 858–865. Citeseer, 2002.

25 Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of generalized vertex cover problems. In *Workshop on Algorithms and Data Structures*, pages 36–48. Springer, 2005.

26 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.

27 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

28 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–192. IEEE, 2022.

29 Michael Lampis. Parameterized approximation schemes using graph widths. In *International Colloquium on Automata, Languages, and Programming*, pages 775–786. Springer, 2014.

30 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2181–2200. SIAM, 2020.

31 Pasin Manurangsi and Prasad Raghavendra. A Birthday Repetition Theorem and Complexity of Approximating Dense CSPs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 78:1–78:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.78.

32 Venkatesh Raman, Saket Saurabh, and Sriganesh Srihari. Parameterized algorithms for generalized domination. In *International Conference on Combinatorial Optimization and Applications*, pages 116–126. Springer, 2008.

33 Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.

34 Barna Saha and Samir Khuller. Set cover revisited: Hypergraph cover with hard capacities. In *International Colloquium on Automata, Languages, and Programming*, pages 762–773. Springer, 2012.

35 Jia-Yau Shiau, Mong-Jen Kao, Ching-Chi Lin, and DT Lee. Tight approximation for partial vertex cover with hard capacities. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

36 Sam Chiu-wai Wong. Tight algorithms for vertex cover with hard capacities on multigraphs and hypergraphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2626–2637. SIAM, 2017.

A Proof Sketch of Theorem 5

It is easy to see that $|R_\alpha| \leq n^{O(w)}$ for all α. And one execution of a recursive rule takes time at most polynomial of the size of some R_α. Thus the total running time is $n^{O(w)} \cdot O(w^2 \log n) = n^{O(w)}$.

To prove the correctness, we need to show the record sets computed by the recursive rules satisfy the expected properties. We use induction. Leaf nodes are trivial to verify. Fix a node α, assume that for every node descendent to it, the corresponding record set is correctly

computed. The proof then contains the 'if' part and the 'only if' part. For the 'if' part we have some O, (d,k) satisfying the expected properties and aim to prove (d,k) is included into R_α by the recursive rules. The framework is to extract O_1 and (d_1,k_1) (and O_2, (d_2,k_2) for join nodes) satisfying the expected properties for the child node(s) and shows that (d,k) will be add into R_α because of (d_1,k_1) (and (d_2,k_2)). By induction, the extracted record will be included by the algorithm because they satisfy the expected properties, so (d,k) will also be included. For the 'only if' part we have (d,k) included and aim to prove the existence of a satisfying O. The framework is to take the record(s) based on which (d,k) is added. By induction, the record(s) we take has corresponding orientation(s) that satisfies the expected properties. We build O according to this(these) orientation(s).

B Proof of Theorem 7

Before the main proof, we prove Lemma 8 and Lemma 9. Remember that Lemma 8 states that if $(d,k) \in R_\alpha$ then $(d',k') \in R_\alpha$ for all (d',k') with $d(v) \geq d'(v), \forall v \in X_\alpha$ and $k \leq k' \leq |Y_\alpha|$; Lemma 9 states that $[a'+b']_\epsilon \sim_{\epsilon_{h+1}} (a+b)$ for $a,b,a',b' \in \mathbb{R}$ satisfying $a' \sim_{\epsilon_h} a$, $b' \sim_{\epsilon_h} b$ for $\epsilon_h \in [0,1]$.

Proof (Lemma 8). Let O be the orientation for (d,k). For each v, we arbitrarily select $d(v) - d'(v)$ out neighbors of v and reverse each edge between one selected neighbor and v. Let the obtained orientation be O_1. We show that O_1 and (d',k') satisfies the properties. (1) and (3) are trivial. To see (2), observe that $D^-_{O_1}(v) \leq D^-_O(v)$ for all $v \in Y_\alpha$. ◀

Proof (Lemma 9). $a'+b' \in [a/(1+\epsilon_h)+b/(1+\epsilon_h), a(1+\epsilon_h)+b(1+\epsilon_h)]$, that is, $(a'+b') \sim_{\epsilon_h} (a+b)$. As $[a'+b']_\epsilon \sim_\epsilon (a'+b')$, we have $\max\{[a'+b']_\epsilon/(a+b), (a+b)/[a'+b']_\epsilon\} \leq (1+\epsilon)(1+\epsilon_h) = 1+\epsilon_{h+1}+\epsilon_h\epsilon-\epsilon \leq 1+\epsilon_{h+1}$. Thus $[a'+b']_\epsilon \sim_{\epsilon_{h+1}} (a+b)$. ◀

In the following we start the main proof. Leaf nodes satisfy property (A) and (B) since $R_\alpha = \hat{R}_\alpha$ for a leaf node α. Fix a node α of height h, by induction, we assume that every node descendent to α satisfies (A) and (B). Now we prove α satisfies both (A) and (B).

B.1 Proof of (A)

Recall that we have some $(d,k) \in R_\alpha$ now and we aim to show the existence of some $(\hat{d},\hat{k}) \in \hat{R}_\alpha$ which is h-close to (d,k). The case for leaf node is trivial. There are three other cases:

Introducing v Node. Suppose α is an introducing v node and α_1 is its child, then we have a certificate $(d_1,k_1) \in R_{\alpha_1}$, where $d_1 = d \setminus v$, $k_1 = k$. By the induction hypothesis, there exists a record $(\hat{d}_1,\hat{k}_1) \in \hat{R}_{\alpha_1}$ which is $(h-1)$-close to (d_1,k_1). By the recursive algorithm for $\hat{R}$, there exists $(\hat{d},\hat{k}) \in \hat{R}_\alpha$, where $\hat{d} \setminus v = \hat{d}_1, \hat{d}(v) = 0$ and $\hat{k} = \hat{k}_1$. Note that for all $u \in X_\alpha \setminus \{v\}, \hat{d}(u) = \hat{d}_1(u) \sim_{\epsilon_{h-1}} d_1(u) = d(u)$, thus we have $\hat{d}(u) \sim_{\epsilon_h} d(u)$. And $\hat{d}(v) = 0 = d(v)$. Since $\hat{k} = \hat{k}_1 \sim_{\delta_{h-1}} k_1 = k$, we get $k \sim_{\delta_h} \hat{k}$. So $(\hat{d},\hat{k})$ is h-close to (d,k).

Join Node. If α is a join node with children α_1 and α_2, then we have a certificate $(d_1,k_1) \in R_{\alpha_1}$ and $(d_2,k_2) \in R_{\alpha_2}$, where for all $v \in X_\alpha, d_1(v) + d_2(v) = d(v)$ and $k_1 + k_2 = k$. By the induction hypothesis, there exist $(\hat{d}_1,\hat{k}_1) \in \hat{R}_{\alpha_1}$ and $(\hat{d}_2,\hat{k}_2) \in \hat{R}_{\alpha_2}$ which are $(h-1)$-close to (d_1,k_1) and (d_2,k_2) respectively. Note that $(\hat{d}_1,\hat{k}_1),(\hat{d}_2,\hat{k}_2)$ is a valid certificate, so there exists $(\hat{d},\hat{k}) \in \hat{R}_\alpha$, where for all $v \in X_\alpha$, $\hat{d}(v) = [\hat{d}_1(v) + \hat{d}_2(v)]_\epsilon$ and $\hat{k} = \hat{k}_1 + \hat{k}_2$. By Lemma 9, for all $v \in X_\alpha$, $\hat{d}(v) \sim_{\epsilon_h} d(v)$ and $\hat{k} \sim_{\delta_h} k$.

Forgetting Node. If α is a forgetting v node with child α_1, then we have a certificate $(d_1, k_1) \in R_{\alpha_1}$ which satisfies one of the following conditions:

(1) $d_1(v) = |N(v) \cap Y_\alpha|, d_1 \setminus v = d$ and $k_1 = k$.

(2) There exist $\Delta(v) \subseteq N(v) \cap X_\alpha$ and $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$ such that for all $u \in \Delta(v), d_1(u) = d(u) - 1$ and for all $u \in X_{\alpha_1} \setminus (\Delta(v) \cup \{v\}), d_1(u) = d(u), d_1(v) = A$ and $k_1 = k - 1$.

Notice that these two conditions just correspond to the recursive rules with the same index. By the induction hypothesis, there exists an approximate counterpart of the certificate. To be specific, there exists $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$ which is $(h-1)$-close to (d_1, k_1). Consider two sub-cases:

Type (1) certificate. As $(\hat{d}_1, \hat{k}_1)$ is $(h-1)$-close to (d_1, k_1) and $d_1(v) = |N(v) \cap Y_\alpha|$, we have $\hat{d}_1(v) \sim_{\epsilon_{h-1}} |N(v) \cap Y_\alpha|$, which means (1a) is satisfied. Let $(d_t, |Y_{\alpha_1}|)$ be the tested pair in (1b). By the definition of $(d_t, |Y_{\alpha_1}|)$, for all $u \in X_{\alpha_1} \setminus \{v\}, d_t(u) = \lceil \hat{d}_1(u)/(1+\epsilon_{h-1}) \rceil \leq \lceil (1+\epsilon_{h-1}) d_1(u)/(1+\epsilon_{h-1}) \rceil = d_1(u)$, and $d_t(v) = d_1(v) = |N(v) \cap Y_\alpha|$. Also observe that $k_1 \leq |Y_{\alpha_1}|$. Thus by Lemma 8, $(d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, which means (1b) is satisfied. As (1a), (1b) are satisfied, there exists $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$, where $\hat{d} = \hat{d}_1 \setminus v, \hat{k} = \hat{k}_1$. Finally, observe that for all $u \in X_\alpha, d(u) = d_1(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u) = \hat{d}(u)$. $k = k_1 \sim_{\delta_{h-1}} \hat{k}_1 = \hat{k}$. So (d, k) and $(\hat{d}, \hat{k})$ are h-close.

Type (2) certificate. As $(\hat{d}_1, \hat{k}_1)$ is $(h-1)$-close to (d_1, k_1) and $d_1(v) = A$, we have $\hat{d}_1(v) \geq A/(1+\epsilon_{h-1})$, which means (2a) is satisfied. Let $(d_t, |Y_{\alpha_1}|)$ be the tested pair in (2b), i.e. for all $u \in X_{\alpha_1} \setminus \{v\}, d_t(u) = \lceil \hat{d}_1(u)/(1+\epsilon_{h-1}) \rceil$ and $d_t(v) = A$. Similarly we have that $d_1(u) \geq d_t(u)$ for all $u \in X_\alpha$ while $k_1 \leq |Y_{\alpha_1}|$. Thus by Lemma 8, $(d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, which means (2b) is satisfied. As (2a), (2b) are satisfied, there exists $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$, where $\hat{d}(u) = [\hat{d}_1(u) + 1]_\epsilon$ for all $u \in X_\alpha \setminus \Delta(v)$, $\hat{d}(u) = \hat{d}_1(u)$ for all $u \in \Delta(v)$, and $\hat{k} = \hat{k}_1 + 1$. For each $u \in \Delta(v)$, $d(u) = d_1(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u) = \hat{d}(u)$; for all $u \in X_\alpha \setminus \Delta(v)$, $d(u) \sim_{\epsilon_h} \hat{d}(u)$ by Lemma 9; $k - 1 = k_1 \sim_{\delta_{h-1}} \hat{k}_1 = \hat{k} - 1$ and thus $k \sim_{\delta_h} \hat{k}$. So (d, k) and $(\hat{d}, \hat{k})$ are h-close.

B.2 Proof of (B)

Now we have some $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$ and we aim to show the existence of some $(d, k) \in R_\alpha$ which is h-close to $(\hat{d}, \hat{k})$.

Introducing v Node. Suppose α is an introducing v node with α_1 as its child, then by the the recursive rules we have a certificate $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$, where $\hat{d}_1 = \hat{d} \setminus v$, $\hat{k}_1 = \hat{k}$. By induction hypothesis, there exists $(d_1, k_1) \in R_{\alpha_1}$ which is $(h-1)$-close to $(\hat{d}_1, \hat{k}_1)$. (d_1, k_1) is a valid certificate, so there exists $(d, k) \in R_\alpha$, where $d \setminus v = d_1, d(v) = 0$ and $k = k_1$. For all $u \in X_\alpha \setminus \{v\}, d(u) = d_1(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u) = \hat{d}(u)$ so $\hat{d}(u) \sim_{\epsilon_h} d(u)$; $\hat{d}(v) = 0 = d(v)$; $k = k_1 \sim_{\delta_{h-1}} \hat{k}_1 = \hat{k}$, so $k \sim_{\delta_h} \hat{k}$.

Join Node. If α is a join node with α_1 and α_2 as its children, then we have a certificate $(\hat{d}_1, \hat{k}_1) \in \hat{R}_\alpha, (\hat{d}_2, \hat{k}_2) \in \hat{R}_{\alpha_2}$, where for all $v \in X_\alpha, [\hat{d}_1(v) + \hat{d}_2(v)]_\epsilon = \hat{d}(v)$ and $\hat{k}_1 + \hat{k}_2 = \hat{k}$. By induction hypothesis, there exist $(d_1, k_1) \in R_{\alpha_1}, (d_2, k_2) \in R_{\alpha_2}$ which are $(h-1)$-close to $(\hat{d}_1, \hat{k}_1)$ and $(\hat{d}_2, \hat{k}_2)$ respectively. Since $(d_1, k_1), (d_2, k_2)$ is a valid certificate, we have there exists $(d, k) \in R_\alpha$, where for all $v \in X_\alpha, d(v) = d_1(v) + d_2(v)$ and $k = k_1 + k_2$. By Lemma 9, for all $v \in X_\alpha, d(v) \sim_{\epsilon_h} \hat{d}(v)$. And $k \sim_{\delta_h} \hat{k}$.

Forgetting v Node. If α is a forgetting v node, then we have a certificate $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$ and a tested pair $(d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$ in (1b) or (2b) with one of the following types:

(1) $\hat{d}_1(v) \sim_{\epsilon_{h-1}} |N(v) \cap Y_\alpha|; \hat{d}_1 \setminus v = \hat{d}$; $\hat{k}_1 = \hat{k}$; $d_t(v) = |N(v) \cap Y_\alpha|$;

(2) there exists $\Delta(v) \subseteq N(v) \cap X_\alpha$ and $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$ such that for all $u \in \Delta(v), \hat{d}(u) = [\hat{d}_1(u) + 1]_\epsilon$; for all $u \in X_{\alpha_1} \setminus \Delta(v) \cup \{v\}, \hat{d}_1(u) = \hat{d}(u); \hat{d}_1(v) \geq A/(1+\epsilon_{h-1}); \hat{k}_1 = \hat{k} - 1; d_t(v) = A$.

In both types, for all $u \in X_{\alpha_1} \setminus \{v\}, d_t(u) = \lceil \hat{d}_1(u)/(1+\epsilon_{h-1}) \rceil$. Notice that these two types just correspond to the recursive rules with the same index. By induction hypothesis, there exists $(d_1, k_1) \in R_{\alpha_1}$ which is $(h-1)$-close to $(\hat{d}_1, \hat{k}_1)$. By the definition of $(h-1)$-closeness we have that for every $u \in X_{\alpha_1} \setminus \{v\}, d_1(u) \geq \lceil \hat{d}_1(u)/(1+\epsilon_{h-1}) \rceil = d_t(u)$. Consider the two cases:

Type (1) certificate and tested pair. In this case $d_t(v) = |N(v) \cap Y_\alpha|$ and $\hat{d}_1(v) \sim_{\epsilon_{h-1}} |N(v) \cap Y_\alpha|$. Notice that for all $u \in X_{\alpha_1} \setminus \{v\}, d_t(u) = \lceil \hat{d}_1(u)/(1+\epsilon_{h-1}) \rceil \leq d_1(u)$. Consider the pair (d_t, k_1^*) where $k_1^* = k_1 + |N(v) \cap Y_\alpha| - d_1(v)$. As $(d_1, k_1), (d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, by Lemma 8 and 11, we have $(d_t, k_1^*) \in R_{\alpha_1}$. This is a valid certificate as $d_t(v) = |N(v) \cap Y_\alpha|$. So there exists $(d, k) \in R_\alpha$, where $d = d_t \setminus v$ and $k = k_1^*$.

Then we show that (d, k) is h-close to $(\hat{d}, \hat{k})$. Notice that $\hat{k} = \hat{k}_1 \sim_{\delta_{h-1}} k_1, k = k_1^* = k_1 + |N(v) \cap Y_\alpha| - d_1(v)$. As $d_1(v) \sim_{\epsilon_{h-1}} \hat{d}_1(v)$, thus $d_1(v) \geq |N(v) \cap Y_\alpha|/(1+\epsilon_{h-1})^2$, thus we have that $|N(v) \cap Y_\alpha| - d_1(v) \leq ((1+\epsilon_{h-1})^2 - 1) d_1(v) \leq 3\epsilon_{h-1} k_1$. Notice that $d_1(v) \leq k_1$ by Lemma 10. So $k \sim_{3\epsilon_{h-1}} k_1 \sim_{\delta_{h-1}} \hat{k}_1 = \hat{k}$. As $(1+3\epsilon_{h-1})(1+\delta_{h-1}) = 1 + (4h+6)(h-1)\epsilon + 24h(h-1)^2\epsilon^2 \leq 1 + 4h(h+1)\epsilon$, we have $\hat{k} \sim_{\delta_h} k$.

For all $u \in X_\alpha$, we just have $d(u) = d_t(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u) = \hat{d}(u)$.

Type (2) certificate and tested pair. In this case, there exists $\Delta(v) \subseteq N(v) \cap X_\alpha$ and $A \in [|N(v) \cap Y_\alpha| - c(v) + |\Delta(v)|, |N(v) \cap Y_\alpha|]$ such that $d_t(v) = A$. Still we have that for all $u \in X_{\alpha_1} \setminus \{v\}, d_t(u) \leq d_1(u)$. Let $k_1^* := k_1 + \max\{0, A - d_1(v)\}$. As $(d_1, k_1), (d_t, |Y_{\alpha_1}|) \in R_{\alpha_1}$, by Lemma 8 and 11, we have $(d_t, k_1^*) \in R_{\alpha_1}$. This is a valid certificate as $d_t(v) = A$. So there exists $(d, k) \in R_\alpha$, where for all $u \in X_\alpha \setminus \Delta(v), d(u) = d_t(u)$, for all $u \in \Delta(v), d(u) = d_t(u) + 1$ and $k = k_1^* + 1$.

We use the same idea to show $\hat{k} \sim_{\delta_h} k$. Still, we have $k_1 \geq d_1(v) \geq A/(1+\epsilon_{h-1})^2$. So $k_1^* = k_1 + \max\{0, A - d_1(v)\} \leq 3\epsilon_{h-1} k_1$ and obviously, $k_1^* \geq k_1$. So $k_1^* \sim_{3\epsilon_{h-1}} k_1$. As $\hat{k} - 1 = \hat{k}_1 \sim_{\delta_{h-1}} k_1$, we have $\hat{k} - 1 \sim_{\delta_h} k_1^* = k - 1$. Thus $\hat{k} \sim_{\delta_h} k$.

For all $u \in X_\alpha \setminus \Delta(v)$, we have $d(u) = {d_1}^*(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u) = \hat{d}$. For all $u \in \Delta(v)$, we have $d(u) - 1 = {d_1}^*(u) \sim_{\epsilon_{h-1}} \hat{d}_1(u)$ and $\hat{d}(u) = [\hat{d}_1(u) + 1]_\epsilon$, by Lemma 9 we have $d(u) \sim_{\epsilon_h} \hat{d}(u)$.

▶ Remark. The above proof actually provides the intuition of how to modify our algorithm so that it outputs a solution of size at most $(1+\delta_{h_0})^2 OPT$. The idea is to, for all $\alpha \in V(T)$ and all $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$, keep track of an exact h-close record (d, k) of $(\hat{d}, \hat{k})$ and its corresponding orientation i.e. an orientation O with which (d, k) satisfies the expected properties. Still, this is done by a bottom-up dynamic programming. Fix a non-leaf node α, suppose that for all its children, this has been done. Now suppose we want to find that orientation for a record $(\hat{d}, \hat{k}) \in \hat{R}_\alpha$. According to the recursive rules, there exists $(\hat{d}_1, \hat{k}_1) \in \hat{R}_{\alpha_1}$ (and $(\hat{d}_2, \hat{k}_2) \in \hat{R}_{\alpha_2}$ for join nodes) from which we construct $(\hat{d}_1, \hat{k}_1)$. Proof of (B) in fact shows that if the exact $h-1$-close exact counterpart and the corresponding orientation has been stored, then we can construct the h-close record $(d, k) \in R_\alpha$ and its corresponding orientation. Notice that if α is the forgetting node we may need Lemma 11 to prove the existence of such (d, k). But fortunately, Lemma 11 is also constructive.

C Proof of Theorem 14

We first prove that for any bag X_α in a tree decomposition for a graph $G = (V, E)$, vertex sets Y_α and $V \setminus V_\alpha$ are disconnected in $G[V \setminus X_\alpha]$ i.e. X_α separates $V \setminus X_\alpha$ into two disconnected parts Y_α and $V \setminus V_\alpha$. Assume they are connected, then there exists $u \in Y_\alpha$ and $v \in V \setminus V_\alpha$

such that $(u, v) \in E$. So there exists some bag containing both u and v. This implies that the nodes whose assigned bags containing u or v forms a subtree in the tree decomposition. However, X divides apart some nodes whose assigned bags containing u or v, a contradiction.

Since $(T, \mathcal{X})$ is a tree decomposition for $G_I[V_I \setminus D]$, a corollary is that for any node $\alpha \in V(T)$, $X_\alpha \cup D$ separates $V_I \setminus (D \cup X_\alpha)$ into disconnected parts Y_α and $V_I \setminus (V_\alpha \cup D)$.

Now we analyze Algorithm 1. We use induction. Firstly let's consider basic cases. If (I, D) has a minimum solution of size at most l, then the algorithm returns at line 8 an optimal solution. If (I, D) contains no solution, which is equivalent to V_I is not a solution due to monotonicity, then any leaf node is not l-good since $Y_{\alpha'} = \emptyset$ for a leaf node α' and the algorithm returns at line 12. So in these cases, the algorithm is correct. In the remaining case, the algorithm picks a node α which is not l-good at line 10, then it adds some vertices to the final output and creates a new instance to make a recursive call. Since α is the node which is not l-good node with minimum height, its children are all l-good. Let the optimal solution for (I, D) be S^*. Let $S := Solve((I, D \cup F), (T', \mathcal{X}), l)$ and let S' denote the optimal solution for $(I, D \cup F)$.

▶ **Lemma 18.** *As the problem is monotone and splittable, we have the following:*

(i) *$S^* \cap Y_\alpha$ is a solution for $(I, V_I \setminus Y_\alpha)$.*

(ii) *For all α_c a child of α, $S^* \cap Y_{\alpha_c}$ is a solution for $(I, V_I \setminus Y_{\alpha_c})$;*

(iii) *$S^* \setminus F$ is a solution for $(I, D \cup F)$;*

(iv) *$E' \cup S$ is a solution for (I, D).*

Proof.

(i) By the definition of partial instances, $S^* \cup D$ is a solution for I. By monotonicity, $S^* \cup D \cup (V_I \setminus Y_\alpha) = S^* \cap Y_\alpha \cup (V_I \setminus Y_\alpha)$ is also a solution for I. So $S^* \cap Y_\alpha$ is a solution for $(I, V_I \setminus Y_\alpha)$ according to the definition of partial solution.

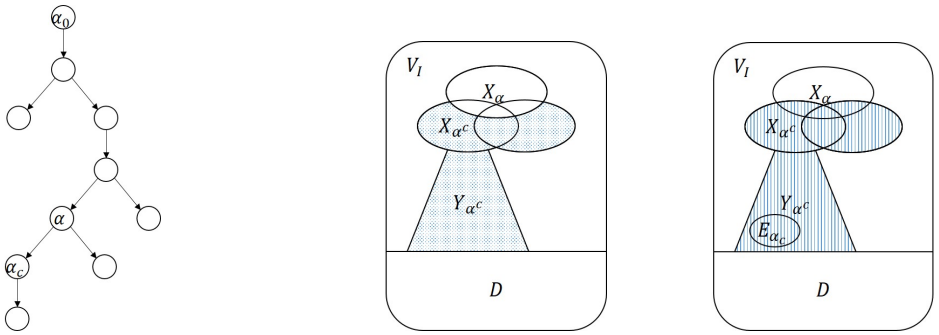
(ii) Similarly as above, by monotonicity, $S^* \cup D \cup (V_I \setminus Y_{\alpha_c}) = S^* \cap Y_{\alpha_c} \cup (V_I \setminus Y_{\alpha_c})$ is also a solution for I. So $S^* \cap Y_{\alpha_c}$ is a solution for $(I, V_I \setminus Y_{\alpha_c})$.

(iii) By monotonicity, $S^* \cup D \cup F$ is also a solution for I. So $S^* \setminus F$ is a solution for $(I, D \cup F)$.

(iv) We need to use the property that Φ is splittable. By the algorithm, $E' = \bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c} \cup \bigcup_{\alpha_c \in N_\alpha^-} E_{\alpha_c}$ and $F = \bigcup_{\alpha_c \in N_\alpha^-} V_{\alpha_c}$. Let X' denote $\cup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}$. To use the property that Φ is splittable, observe that $D \cup X'$ is a separator. Each Y_{α_c} is an isolated part (not connected to the remaining graph) in $G_I[V_I \setminus (D \cup X')]$. The remaining part in $G_I[V_I \setminus (D \cup X')]$ is thus isolated and it is $V_I \setminus (D \cup X' \cup \bigcup_{\alpha_c \in N_\alpha^-} Y_{\alpha_c}) = V_I \setminus (D \cup F)$. Because each E_{α_c} is a solution for $(I, V_I \setminus Y_{\alpha_c})$, and by induction hypothesis, S is a solution for $(I, D \cup F)$, we get that Φ is splittable implies $X' \cup D \cup S \cup \bigcup_{\alpha_c \in N_\alpha^-} E_{\alpha_c} = E' \cup D \cup S$ is a solution for I. So $E' \cup S$ is a solution for (I, D). ◀

By induction we assume that $|S| \leq (1 + (w+1)/(l+1))|S'|$. The approximation ratio is

$$\frac{|S \cup E'|}{|S^*|} \leq \frac{|S| + \sum_{\alpha_c \in N_\alpha^-} |E_{\alpha_c}| + |\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|}{|S^* \cap F| + |S^* \setminus F|}.$$



(a) T in a tree decomposition $(T, \mathcal{X})$.

(b) The vertex sets about α and α_c. Dotted part is Y_α.

(c) E_{α_c} is added, and F is the lined part.

Figure 1 Venn diagram of sets defined in Algorithm 1.

Since $|S|/|S^* \setminus F| \leq |S|/|S'| \leq 1 + (w+1)/(l+1)$, we only need to show $(\sum_{\alpha_c \in N_\alpha^-} |E_{\alpha_c}| + |\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|)/|S^* \cap F| \leq 1 + (w+1)/(l+1)$. Notice that by the definition, $Y_\alpha \subseteq F$. Since α is not l-good, (i) implies that $|S^* \cap F| \geq |S^* \cap Y_\alpha| \geq l+1$. By (ii), for all $\alpha_c \in N_\alpha^-$, $|E_{\alpha_c}| \leq |S^* \cap Y_{\alpha_c}|$. We have

$$
\begin{aligned}
&\frac{\sum_{\alpha_c \in N_\alpha^-} |E_{\alpha_c}| + |\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|}{|S^* \cap F|} \\
=&\frac{\sum_{\alpha_c \in N_\alpha^-} |E_{\alpha_c}|}{|S^* \cap F|} + \frac{|\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|}{|S^* \cap F|} \\
\leq&\frac{\sum_{\alpha_c \in N_\alpha^-} |E_{\alpha_c}|}{\sum_{\alpha_c \in N_\alpha^-} |S^* \cap Y_{\alpha_c}|} + \frac{|\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|}{|S^* \cap F|} \text{ (Y_{α_c}'s are disjoint subsets of F)} \\
\leq&1 + \frac{|\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|}{|S^* \cap F|} \text{ (By (ii) and the definition of E_{α_c})} \\
\leq&1 + \frac{|\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|}{l+1} \text{ (By $|S^* \cap F| \geq l+1$).}
\end{aligned}
$$

In a nice tree decomposition, the only case that $|N_\alpha^-| > 1$ is that α is a join node, however in this case, the bags of its two children are the same. So $|\bigcup_{\alpha_c \in N_\alpha^-} X_{\alpha_c}|/(l+1) + 1 \leq (w+1)/(l+1) + 1$. The approximation ratio follows. Each time we make a recursive call, the optimal solution size for the current instance decreases by at least 1. It follows that the algorithm makes at most $O(n)$ recursive calls, so the running time is $f(l, w, n)n^{O(1)}$. And thus Theorem 14 is proved.

Improved Approximation for Two-Dimensional Vector Multiple Knapsack

Tomer Cohen
Computer Science Department, Technion, Haifa, Israel

Ariel Kulik
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Hadas Shachnai
Computer Science Department, Technion, Haifa, Israel

Abstract

We study the UNIFORM 2-DIMENSIONAL VECTOR MULTIPLE KNAPSACK (2VMK) problem, a natural variant of MULTIPLE KNAPSACK arising in real-world applications such as virtual machine placement. The input for 2VMK is a set of items, each associated with a 2-dimensional *weight* vector and a positive *profit*, along with m 2-dimensional bins of uniform (unit) capacity in each dimension. The goal is to find an assignment of a subset of the items to the bins, such that the total weight of items assigned to a single bin is at most one in each dimension, and the total profit is maximized.

Our main result is a $(1 - \frac{\ln 2}{2} - \varepsilon)$-approximation algorithm for 2VMK, for every fixed $\varepsilon > 0$, thus improving the best known ratio of $(1 - \frac{1}{e} - \varepsilon)$ which follows as a special case from a result of [Fleischer at al., MOR 2011]. Our algorithm relies on an adaptation of the Round&Approx framework of [Bansal et al., SICOMP 2010], originally designed for set covering problems, to maximization problems. The algorithm uses randomized rounding of a configuration-LP solution to assign items to $\approx m \cdot \ln 2 \approx 0.693 \cdot m$ of the bins, followed by a reduction to the (1-dimensional) MULTIPLE KNAPSACK problem for assigning items to the remaining bins.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases vector multiple knapsack, two-dimensional packing, randomized rounding, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.20

Related Version *Full Version*: https://arxiv.org/abs/2307.02137

Funding *Ariel Kulik*: Research supported by the European Reseach Concil (ERC) consolidator grant no. 725978 SYSTEMATICGRAPH.

1 Introduction

The KNAPSACK problem and its variants have attracted much attention in the past four decades, and have been instrumental in the development of approximation algorithms. In this paper we study a variant of KNAPSACK which uses components of two well studied knapsack problems: MULTIPLE KNAPSACK and 2-DIMENSIONAL KNAPSACK.

An instance of uniform MULTIPLE KNAPSACK consists of a set I of items of non-negative profits and weights in $[0, 1]$, as well as m uniform (unit size) bins. We seek a subset of the items of maximal total profit which can be packed in the m bins. An instance of 2-DIMENSIONAL KNAPSACK is a set I of items, each has a 2-dimensional weight in $[0, 1]^2$, and a non-negative profit. The objective is to find a subset of the items whose total weight is at most one in each dimension, such that the total profit is maximized. We study a variant of uniform MULTIPLE KNAPSACK where each bin is a 2-dimensional knapsack, thus generalizing both problems.

Formally, an instance of UNIFORM 2-DIMENSIONAL MULTIPLE KNAPSACK (2VMK) is a tuple $\mathcal{I} = (I, w, p, m)$, where I is a set of items, $w : I \to [0,1]^2$ is a 2-dimensional weight function, $p : I \to \mathbb{R}_{\geq 0}$ is a profit function, and m is the number of bins. For any $k \in \mathbb{N}$, let $[k] = \{1, 2, \ldots, k\}$. A *solution* for the instance (I, w, p, m) is a collection of subsets of items $S_1, \ldots, S_m \subseteq I$ such that $w(S_b) = \sum_{i \in S_b} w(i) \leq (1,1)$ for all $b \in [m]$.[1] Our objective is to find a solution $S_1, \ldots, S_m$ which maximizes the total profit, given by $p\left(\bigcup_{b \in [m]} S_b\right)$.

A natural application of 2VMK arises in the cloud computing environment. Consider a data center consisting of m hosts (physical machines). Each host has an available amount of processing power (CPU) and limited memory. For simplicity, these amounts can be scaled to one unit. The data center administrator has a queue of client requests to assign *virtual machines* (VMs) to the hosts.[2] Each VM has a demand for processing power and memory, and its execution is associated with some profit. The administrator needs to assign a subset of the VMs to the hosts, such that the total processing power and memory demands on each host do not exceed the available amounts, and the profit gained from the VMs is maximized (see [5] for other optimization objectives in this setting). Another application of 2VMK comes from spectrum allocation in cognitive radio networks [18].

Our goal is to develop an efficient polynomial-time approximation algorithm for 2VMK. Let $\alpha \in (0, 1]$ be a constant. An algorithm $\mathcal{A}$ is an α-approximation algorithm for 2VMK if for any instance $\mathcal{I}$ of 2VMK it returns in polynomial-time a solution of profit at least $\alpha \cdot \mathrm{OPT}(\mathcal{I})$, where $\mathrm{OPT}(\mathcal{I})$ is the optimal profit for $\mathcal{I}$. A *polynomial-time approximation scheme* (PTAS) is an infinite family $\{A_\varepsilon\}$ of $(1-\varepsilon)$-approximation algorithms, one for each $\varepsilon > 0$. A weaker notion is that of a *randomized* α-approximation algorithm, where the algorithm always returns a solution, but the profit is at least $\alpha \cdot \mathrm{OPT}(\mathcal{I})$ with some constant probability.

As the classic MULTIPLE KNAPSACK problem admits an *efficient PTAS* (EPTAS), even for instances with arbitrary bin capacities [10, 11], and the single bin problem, i.e., 2-DIMENSIONAL KNAPSACK has a PTAS [9], a natural question is whether 2VMK admits a PTAS as well. By a simple reduction from 2-DIMENSIONAL VECTOR BIN PACKING (2VBP), we show that such a PTAS is unlikely to exist.[3]

▶ **Theorem 1.** *Assuming* P $\neq$ NP *there is no PTAS for* 2VMK.

Thus, we focus on deriving the best constant factor approximation for the problem. For any $\varepsilon > 0$, a randomized $(1 - e^{-1} - \varepsilon) \approx 0.632$-approximation algorithm for 2VMK follows from a result of [8], as a special case of the SEPARABLE ASSIGNMENT PROBLEM (SAP). Our main result is an improved approximation ratio for the problem.

▶ **Theorem 2.** *For every fixed* $\varepsilon > 0$*, there is a randomized* $\left(1 - \frac{\ln 2}{2} - \varepsilon\right) \approx 0.653$*-approximation algorithm for* 2VMK*.*

1.1 Prior Work

The special case of 2VMK with a single bin, i.e., 2-DIMENSIONAL KNAPSACK, admits a PTAS due to Frieze and Clarke [9]. As shown in [14], an EPTAS for the problem is unlikely to exist. The first PTAS for MULTIPLE KNAPSACK was presented by Chekuri and Khanna [6] who

[1] We say that $(a_1, a_2) \leq (b_1, b_2)$ if $a_1 \leq b_1$ and $a_2 \leq b_2$.
[2] This is also known as virtual machine *instantiation* [5].
[3] We give the proof of Theorem 1 in Section 4.

also showed the problem is strongly NP-hard. The PTAS was later improved to an EPTAS by Jansen [10, 11]. For comprehensive surveys of known results on knapsack problems, see, e.g., [12, 4].

Both Multiple Knapsack and 2VMK are special cases of the Separable Assignment Problem (SAP) studied in [8]. The input for SAP is a set of items I and m bins. Each item $i \in I$ has a profit $p_{i,j}$ if assigned to bin $j \in [m]$. Each bin $j \in [m]$ is associated with a collection of feasible assignments $\mathcal{F}_j \subseteq 2^I$. The feasible assignments are hereditary, that is, if $S \in \mathcal{F}_j$ and $T \subseteq S$ then $T \in \mathcal{F}_j$ for all $T \subseteq S$. The feasible assignments are given implicitly via a β-optimization oracle which, given a value function $v : I \to \mathbb{R}_{\geq 0}$ and $j \in [m]$, finds a β-approximate solution for $\max_{S \in \mathcal{F}_j} \sum_{i \in S} v(i)$. A solution for the SAP instance is a tuple of disjoint sets $S_1, \ldots, S_m \subseteq I$ such that $S_j \in \mathcal{F}_j$ for all $j \in [m]$. The objective is to find a solution $S_1, \ldots, S_m$ of maximum profit $\sum_{j=1}^{m} \sum_{i \in S_j} p_{i,j}$. A $(1 - e^{-1}) \cdot \beta$-approximation for SAP was given in [8].

Observe that 2VMK can be cast as SAP by setting $p_{i,j} = p(i)$ for every $(i, j) \in I \times [m]$, and $\mathcal{F}_j = \{S \subseteq I \mid w(S) \leq (1, 1)\}$ for every $j \in [m]$. That is, the profit of item i is $p(i)$ regardless of the bin to which it is assigned, and the feasible assignments of all bins are simply all subsets of items which fit into a bin. For every fixed $\varepsilon > 0$, a $(1 - \varepsilon)$-optimization oracle for the bins can be implemented in polynomial time using the PTAS of [9] for 2-dimensional knapsack. Hence, a $(1 - e^{-1} - \varepsilon)$-approximation for 2VMK follows from the result of [8].

Parameterized algorithms for 2VMK were proposed in [15, 1]. We are not aware of earlier works which directly study 2VMK from approximation algorithms viewpoint.

1.2 Technical Overview

Our algorithm combines the approximation algorithm of Fleischer et al. [8] with a simple reduction of 2VMK to (1-dimensional) Multiple Knapsack.

The algorithm of [8] is based on randomized rounding of a *configuration-LP* solution. Given an instance $\mathcal{I} = (I, w, p, m)$ of 2VMK, let $w_j(i)$ denote the weight of item i in the jth coordinate, for all $i \in I$ and $j \in \{1, 2\}$. Also, given $f : I \to \mathbb{R}^d$ we use the notation $f(S) = \sum_{i \in S} f(i)$ for all $S \subseteq I$. A *configuration* of the instance $\mathcal{I}$ is $C \subseteq I$ such that $w(C) \leq (1, 1)$. Let $\mathcal{C}(\mathcal{I})$ be the set of all configurations of the 2VMK instance $\mathcal{I}$. For any $i \in I$, let $\mathcal{C}(\mathcal{I}, i)$ be the set of configurations containing item i. We often omit $\mathcal{I}$ and use $\mathcal{C}$ and $\mathcal{C}(i)$ when the instance $\mathcal{I}$ is known by context. Observe that a solution for the instance $\mathcal{I}$ is a tuple of m configurations.

Given a 2VMK instance $\mathcal{I} = (I, w, p, m)$, let $\mathbf{x}_C \in \{0, 1\}$ be an indicator for the selection of configuration $C \in \mathcal{C} = \mathcal{C}(\mathcal{I})$ for the solution. In the configuration-LP relaxation of the problem, we have $\mathbf{x}_C \geq 0 \ \forall C \in \mathcal{C}$. Our algorithm initially solves the following.

$$\text{(C-LP)} \qquad \max \quad \sum_{C \in \mathcal{C}} \sum_{i \in C} \mathbf{x}_C \cdot p(i)$$

$$\text{subject to :} \quad \sum_{C \in \mathcal{C}} \mathbf{x}_C \leq m \tag{1}$$

$$\sum_{C \in \mathcal{C}(i)} \mathbf{x}_C \leq 1 \qquad \forall i \in I \tag{2}$$

$$\mathbf{x}_C \geq 0 \qquad \forall C \in \mathcal{C}$$

The next lemma follows from a result of [8].

▶ **Lemma 3.** *There is a PTAS for* C-LP.

Let $\mathbf{x}$ be a solution for C-LP. We say that a random configuration $R \in \mathcal{C}$ is *distributed by* $\mathbf{x}$ if $\Pr[R = C] = \frac{\mathbf{x}_C}{m}$.[4]

To obtain a $\left(\left(1 - e^{-1}\right) \cdot (1 - \varepsilon)\right)$-approximation for 2VMK, the algorithm of [8] finds a $(1 - \varepsilon)$-approximate solution $\mathbf{x}$ for C-LP, and then independently samples m configuration $R_1, \ldots, R_m \in \mathcal{C}$, where each of the configurations R_j is distributed by $\mathbf{x}$. The returned solution is the sampled configurations $R_1, \ldots, R_m$. For simplicity of this informal overview, assume that $\sum_{C \in \mathcal{C}} \sum_{i \in C} \mathbf{x}_C \cdot p(i) \approx \mathrm{OPT}(\mathcal{I})$.

It can be shown that

$$\mathbb{E}\left[p(R_1 \cup \ldots \cup R_j)\right] \approx \left(1 - e^{-\frac{j}{m}}\right) \cdot \sum_{C \in \mathcal{C}} \sum_{i \in C} \mathbf{x}_C \cdot p(i) \approx \left(1 - e^{-\frac{j}{m}}\right) \cdot \mathrm{OPT}(\mathcal{I}) \tag{3}$$

for all $j \in [m]$, and in particular $\mathbb{E}\left[p(R_1 \cup \ldots \cup R_m)\right] \approx \left(1 - e^{-1}\right) \cdot \mathrm{OPT}(\mathcal{I})$. Define

$$q_j = \mathbb{E}\left[p(R_1 \cup \ldots \cup R_j) - p(R_1 \cup \ldots \cup R_{j-1})\right]$$

to be the marginal expected profit of the jth sampled configuration. By (3) we have

$$q_j \approx \left(\left(1 - e^{-\frac{j}{m}}\right) - \left(1 - e^{-\frac{j-1}{m}}\right)\right) \cdot \mathrm{OPT}(\mathcal{I}) \approx \frac{1}{m} \cdot e^{-\frac{j}{m}} \cdot \mathrm{OPT}(\mathcal{I}), \tag{4}$$

where the last estimation follows from $e^{-\frac{j-1}{m}} \approx e^{-\frac{j}{m}} + \frac{1}{m} \cdot e^{-\frac{j}{m}}$ by a Taylor expansion of e^x at $x = -\frac{j}{m}$. Equation (4) implies that the marginal profit from each configuration decreases as the sampling process proceeds. The first sampled configuration has a marginal profit of $\approx \frac{1}{m} \cdot \mathrm{OPT}(\mathcal{I})$, while the marginal profit of the mth configuration is $\approx e^{-1} \cdot \frac{1}{m} \cdot \mathrm{OPT}(\mathcal{I}) \approx 0.367 \cdot \frac{1}{m} \cdot \mathrm{OPT}(\mathcal{I})$.

We note that a simple reduction to Multiple Knapsack can be used to derive a $\left(\frac{1}{2} - \varepsilon\right)$-approximation for 2VMK. An instance $\mathcal{J}$ of uniform multiple knapsack (MK) is a tuple $\mathcal{J} = (I, w, p, m)$, where I is a set of items, $w : I \to [0, 1]$ is a weight function, $p : I \to \mathbb{R}_{\geq 0}$ is a profit function, and m is the number of (unit size) bins. A *configuration* of $\mathcal{J}$ is a subset of items $C \subseteq I$ satisfying $w(C) \leq 1$. We use $\mathcal{C}(\mathcal{J})$ to denote the set of all configurations of $\mathcal{J}$, and sometimes omit $\mathcal{J}$ from this notation if it is known by context. A feasible solution for $\mathcal{J}$ is a tuple of m configurations $C_1, \ldots, C_m \in \mathcal{C}(\mathcal{J})$. The objective is to find a solution $C_1, \ldots C_m$ for which the total profit, given by $p(\bigcup_{t \in [m]} C_t)$, is maximized. We note that uniform MK can be viewed as uniform 1-dimensional vector multiple knapsack.

Let $\mathcal{I} = (I, w, p, m)$ be a 2VMK instance, and define an MK instance $\mathcal{J} = (I, w', p, m)$ where $w'(i) = \max\{w_1(i), w_2(i)\}$ for every $i \in I$. It can be easily shown that $\mathcal{C}(\mathcal{J}) \subseteq \mathcal{C}(\mathcal{I})$, i.e., a configuration of the MK instance is a configuration of the 2VMK instance. Furthermore, every $C \in \mathcal{C}(\mathcal{I})$ can be partitioned into $C_1, C_2 \in \mathcal{C}(\mathcal{J})$ (possibly with $C_1 = \emptyset$). That is, every configuration of the 2VMK instance can be split into two configurations of the MK instance. Therefore $\mathrm{OPT}(\mathcal{J}) \geq \frac{1}{2} \cdot \mathrm{OPT}(\mathcal{I})$, as we can take the optimal solution of the 2VMK instance $\mathcal{I}$, convert its configurations to $2m$ configurations of the MK instance $\mathcal{J}$, and select the m most profitable ones. As MK admits a PTAS [6], we can find a $(1 - \varepsilon)$-approximate solution for $\mathcal{J}$ in polynomial time. This leads to the following algorithm, to which we refer as the *reduction algorithm*. Given the instance $\mathcal{I}$, return a $(1 - \varepsilon)$-approximate solution for $\mathcal{J}$. The above arguments can be used to show that the reduction algorithm is a $\left(\frac{1}{2} - \varepsilon\right)$-approximation algorithm for 2VMK.

[4] W.l.o.g we assume that $\|x\| = \sum_{C \in \mathcal{C}} \mathbf{x}_C = m$.

While this simple $\left(\frac{1}{2}-\varepsilon\right)$-approximation is inferior to the $\left(1-e^{-1}-\varepsilon\right)$-approximation of Fleischer et al. [8], it is still useful for obtaining a better approximation for 2VMK, due to the following property. Let $T \subseteq I$ be a random subset of the items such that $\Pr(i \in T) \leq \alpha$ for all $i \in I$, and T satisfies some concentration bounds. Then, if the reduction algorithm is executed with the 2VMK instance $(I \setminus T, p, w, (1-\alpha)m)$, it returns with high probability a solution of profit at least $\frac{1}{2} \cdot (1-\alpha) \cdot \mathrm{OPT}(\mathcal{I})$. In other words, if every item is removed from the instance with probability at most α, then the algorithm can find a packing into $(1-\alpha) \cdot m$ bins that yields $\frac{1}{2} \cdot (1-\alpha)$ of the original profit $\mathrm{OPT}(\mathcal{I})$. This property resembles the notion of *subset oblivious* algorithms used by [2] as part of their Round&Approx framework for set covering problems. Indeed, we use a subset oblivious algorithm of [2] in our proof of this property.

This suggests the following hybrid algorithm. First, solve C-LP and sample $\ell \approx \alpha m$ configuration $R_1, \ldots, R_\ell$ distributed by the C-LP solution $\mathbf{x}$. Subsequently, define $T = \bigcup_{j=1}^{\ell} R_j$, and use the reduction algorithm to find a solution S for the instance $(I \setminus T, w, p, (1-\alpha)m)$. The algorithm returns $R_1, \ldots, R_\ell$ together with the $m - \ell = (1-\alpha)m$ configurations of S. It can be shown that $\Pr(i \in T) \leq \alpha$ for all T, and that T satisfies the required concentration bounds; therefore, the expected profit of S is $\approx \frac{1}{2} \cdot (1-\alpha) \cdot \mathrm{OPT}(\mathcal{I})$. Since S uses $(1-\alpha)m$ configurations, this can be interpreted as a residual profit of $\frac{1}{2 \cdot m} \cdot \mathrm{OPT}(\mathcal{I})$ per configuration. This property suggests how to select α. We set the value of α such that the marginal profit of the (αm)-th sampled configuration is equal to $\frac{1}{2m} \cdot \mathrm{OPT}(\mathcal{I})$, the marginal profit of a configuration in S. By (4), this is realized for $\alpha = \ln 2$.

We note that this hybrid approach is similar to the Round&Approx framework of Bansal et al. [2], which solves set covering problems by sampling random configurations based on a solution for a configuration-LP, followed by a subset oblivious algorithm which completes the solution. Our algorithm can be viewed as an adaptation of the framework of [2] to knapsack variants. To the best of our knowledge, this is the first application of ideas from [2] to maximization problems.

Our proofs rely on a dimension-free concentration bound for self-bounding functions due to Boucheron et al. [3] (see Section 3.1 for details). While our approach is conceptually similar to the Round&Approx of [2], which uses McDiarmid's bound [16] to show concentration, it appears that McDiarmid's bound does not suffice to guarantee concentration of the profits. The bound of Boucheron et al. was previously used by Vondrák to show concentration bounds for submodular functions [20]. We are not aware of other applications of this bound in the context of combinatorial optimization.

1.3 Organization

In Section 2 we give some definitions and preliminary results. Section 3 presents our approximation algorithm for 2VMK, and in Section 4 we give a proof of APX-hardness (stated in Theorem 1). We conclude in Section 5 with a summary and some directions for future work. Due to space constraints, some of the proofs are given in the full version of the paper [7].

2 Preliminaries

For any function $f : I \to \mathbb{R}^d$ where $d \in \{1, 2\}$ we use the notation $f(A) = \sum_{i \in A} f(i)$. We note that in both MK and 2VMK an item selected for the solution may appear in more than one configuration; however, the profit of each selected item is counted exactly once.

Let $\mathcal{I}$ be an instance of either 2VMK or MK. We say that a subset of items $S \subseteq I$ *can be packed into q bins of $\mathcal{I}$*, for some $1 \leq q \leq m$, if there are configurations $C_1, \ldots, C_q \in \mathcal{C}(\mathcal{I})$ such that $\cup_{t=1}^{q} C_t = S$.

Finally, given $n \in \mathbb{N}$ and an arbitrary set $\mathcal{X}$, define $\mathcal{X}^n$ to be the set of all vectors of dimension n over $\mathcal{X}$; that is, $\mathcal{X}^n = \{(x_1, \ldots, x_n) \mid \forall t \in \{1, \ldots, n\} : \ x_t \in \mathcal{X}\}$.

2.1 Associated Instances

Our approximation algorithm for 2VMK uses as a subroutine an approximation algorithm for MK. To this end, we define for a given 2VMK instance an associated MK instance. Formally, given a 2VMK instance $\mathcal{I} = (I, w, p, m)$, we define the *$k$-associated MK instance* $\mathcal{I}' = (I', w', p', m')$ as follows. The set of items is $I' = I$. The item weights are given by $w'(i) = \max\{w_1(i), w_2(i)\}$, for $1 \leq i \leq n$; the profit of item i is $p'(i) = p(i)$, and the number of (unit size) bins is $m' = k \cdot m$. The next result will be useful in analyzing our algorithm for 2VMK.

▶ **Lemma 4.** *Let $\mathcal{I} = (I, w, p, m)$ be a 2VMK instance, and $\mathcal{I}' = (I', w', p', m')$ its k-associated instance. Then, any set $S \subseteq I$ that can be packed in q bins of $\mathcal{I}$, can be packed in $2 \cdot q$ bins of $\mathcal{I}'$.*

2.2 Restriction to ε-Nice Instances

The correctness proofs for our approach require the number of bins m to be large, and the maximum profit for a single configuration to be relatively small. These properties are essential for the concentration bounds that we use to ensure success with high probability. We show that for instances in which m is small (i.e., bounded by some constant), a simple reduction to d-dimensional knapsack with a matroid constraint yields a PTAS. This allows us to focus on instances with a large number of bins. Furthermore, for such instances we use a simple greedy pre-processing to ensure bounded maximal profit for a single configuration. Thus, we restrict our attention to the following subclass of *ε-nice* instances. Let $\exp^{[k]}(x) = \exp(\exp^{[k-1]}(x))$, for any integer $k \geq 2$, and $\exp^{[1]}(x) = \exp(x)$.

▶ **Definition 5.** *Given $\varepsilon \in (0, 0.01)$, an instance $\mathcal{I} = (I, w, p, m)$ is* ε-nice *if $m \geq \exp^{[3]}(\varepsilon^{-30})$ and $p(C) \leq \varepsilon^{20} \cdot \mathrm{OPT}(\mathcal{I})$ for every $C \in \mathcal{C}$.*[5]

We show that efficient approximation for 2VMK on ε-nice instances yields almost the same approximation ratio for general instances.

▶ **Lemma 6.** *For any $\varepsilon \in (0, 0.01)$ and $\beta \in (0, 1 - \varepsilon)$, if there is polynomial-time β-approximation algorithm for* 2VMK *on ε-nice instances, then there is a polynomial-time $(1 - \varepsilon) \cdot \beta$-approximation algorithm for* 2VMK.

2.3 Two-dimensional Vector Bin Packing

An instance $\mathcal{I}$ of the 2-Dimensional Vector Bin Packing (2VBP) problem is a pair (I, w), where I is a set of n items and $w : I \to [0, 1]^2$ is a two-dimensional weight function. A *solution* for the instance (I, w) is a collection of subsets of items $S_1, \ldots, S_m \subseteq I$ such that $w(S_b) = \sum_{i \in S_b} w(i) \leq (1, 1)$ for all $b = 1, \ldots, m$, and $\bigcup_{b=1}^{m} S_b = I$. The *size* of the solution is m. Our objective is to find a solution of minimum size.

[5] We did not attempt to optimize the constants.

An *asymptotic polynomial-time approximation scheme* (APTAS) is an infinite family $\{A_\varepsilon\}$ of asymptotic $(1-\varepsilon)$-approximation algorithms, one for each $\varepsilon > 0$. Ray [17] showed that 2VBP does not admit an asymptotic approximation ratio better than $\frac{600}{599}$, assuming $P \neq NP$; thus, 2VBP does not admit an APTAS.

3 Approximation Algorithm for ε-Nice Instances

In this section we present an algorithm for ε-nice 2VMK instances. Our algorithm proceeds by initially obtaining an approximate solution $\mathbf{x}$ for C-LP (as given in Section 1.2), and then forming a partial solution by sampling $1 \leq \ell \leq m$ configurations. The remaining $(m-\ell)$ configurations are derived by solving the associated MK instance for the remaining (unassigned) items. The pseudocode of our algorithm is given in Algorithm 1.

Algorithm 1 Approximation Algorithm for ε-nice instances.

configuration : $\varepsilon \in (0, 0.01)$.
input : An *ε-nice* instance $\mathcal{I} = (I, w, p, m)$ of 2VMK.
output : A solution for the instance $\mathcal{I}$.

1: Find a $(1-\varepsilon)$-approximate solution $\mathbf{x}$ for C-LP; let x^* be its value.
2: **for** $t = 1$ *to* $\ell = \lceil m \cdot \ln 2 \rceil$ **do**
| Sample a random configuration R_t distributed by $\mathbf{x}$
end
3: $S \leftarrow I \setminus (\cup_{t\in\{1,\ldots,l\}} R_t)$
4: Let $\mathcal{I}'$ be the 1-associated MK instance of the 2VMK instance $(S, w, p, m-\ell)$.
5: Find a $(1-\varepsilon)$-approximate solution for the MK instance $\mathcal{I}'$; denote the solution by $R_{\ell+1}, \ldots, R_m$.
6: Return $(R_1, \ldots, R_m)$.

Note that, by Lemma 3, Step 1 of Algorithm 1 can be implemented in polynomial time, for any fixed $\varepsilon > 0$. Let $\varepsilon \in (0, 0.01)$. and $\mathcal{I}$ be an ε-nice 2-VMK instance. Consider the execution of Algorithm 1 configured by ε with $\mathcal{I}$ as its input. Let OPT be the set of items selected by an optimal solution for $\mathcal{I}$, and $T = \bigcup_{t\in\{1,\ldots,\ell\}} R_t$ the items selected in Step 2 in Algorithm 1. We use the next lemmas in the analysis of the algorithm. Lemma 7 lower bounds the expected profit of $R_1, \ldots, R_\ell$, the configurations sampled in Step 2 of Algorithm 1. Lemma 8 gives a lower bound on the profit of the MK solution $R_{\ell+1}, \ldots, R_m$ found in Step 5 of Algorithm 1. Lemma 9 lower bounds the profit of the solution returned by the algorithm using the bounds in Lemmas 7 and 8, whose proofs are given in Sections 3.2 and 3.3.

▶ **Lemma 7.** $\Pr\left[p(T) \leq (1 - e^{-\alpha} - 2\cdot\varepsilon)\cdot p(\mathrm{OPT})\right] \leq \exp\left(-\varepsilon^{-7}\right)$.

▶ **Lemma 8.** $\Pr\left[p(\bigcup_{t=\ell+1}^{m} R_t) \leq \left(\frac{1-\alpha}{2} - 3\cdot\varepsilon\right)\cdot p(\mathrm{OPT})\right] \leq \frac{1}{4}$.

▶ **Lemma 9.** *Algorithm 1 returns a solution of profit at least* $\left(1 - \frac{\ln 2}{2} - 5\cdot\varepsilon\right)\cdot p(\mathrm{OPT})$ *with probability at least* $\frac{1}{2}$.

For an event A, let $\bar{A}$ denote the complementary event.

Proof of Lemma 9. Let A be the event "$p(T) > (1-e^{-\alpha}-2\cdot\varepsilon)\cdot p(\mathrm{OPT})$", and B the event "$p\left(\bigcup_{t=\ell+1}^{m} R_t\right) > \left(\frac{1-\alpha}{2} - 3\varepsilon\right)\cdot p(\mathrm{OPT})$". If both A and B occur then Algorithm 1 returns a solution of profit at least

$$p(T) + p\left(\bigcup_{t=\ell+1}^{m} R_t\right) \geq \left(1 - e^{-\alpha} - 2\varepsilon + \frac{1-\alpha}{2} - 3\varepsilon\right) p(\mathrm{OPT}) \quad = \left(1 - \frac{\ln 2}{2} - 5\varepsilon\right) p(\mathrm{OPT}).$$

The inequality holds since both A and B occur. The equality holds since $\alpha = \ln 2$. The probability that A and B occur is given by

$$\Pr[A \cap B] = 1 - \Pr\left[\bar{A} \cup \bar{B}\right] \quad \geq 1 - \left(\Pr\left[\bar{A}\right] + \Pr\left[\bar{B}\right]\right) \geq 1 - \exp\left(-\varepsilon^{-7}\right) - \frac{1}{4} \geq \frac{1}{2}$$

The first inequality follows from the union bound. The second inequality follows from Lemmas 7 and 8, and since $\varepsilon < 0.01$. ◀

3.1 Self-Bounding Functions

Lemmas 7 and 8 we use a concentration bound for *self-bounding functions*.

▶ **Definition 10.** *A non-negative function $f : \mathcal{X}^n \to \mathbb{R}_{\geq 0}$ is called self-bounding if there exist n functions $f_1, \ldots, f_n : \mathcal{X}^{n-1} \to \mathbb{R}$ such that for all $x = (x_1, \ldots, x_n) \in \mathcal{X}^n$,*

$$0 \leq f(x) - f_t(x^{(t)}) \leq 1, \qquad \text{and}$$
$$\sum_{t=1}^{n} \left(f(x) - f_t(x^{(t)})\right) \leq f(x),$$

where $x^{(t)} = (x_1, \ldots, x_{t-1}, x_{t+1}, \ldots, x_n) \in \mathcal{X}^{n-1}$ is obtained by dropping the t-th component of x.

The next result is shown in [3].

▶ **Lemma 11.** *Let $f : \mathcal{X}^n \to \mathbb{R}_{\geq 0}$ be a self-bounding function and let $X_1, \ldots, X_n \in \mathcal{X}$ be independent random variables. Define $Z = f(X_1, \ldots, X_n)$. Then the following holds:*

1. $\Pr[Z \geq \mathbb{E}[Z] + t] \leq \exp\left(-\frac{t^2}{2 \cdot \mathbb{E}[Z] + \frac{t}{3}}\right)$, *for every $t \geq 0$.*
2. $\Pr[Z \leq \mathbb{E}[Z] - t] \leq \exp\left(-\frac{t^2}{2 \cdot \mathbb{E}[Z]}\right)$, *for every $0 < t < \mathbb{E}[Z]$.*

We use the following construction of self-bounding functions several times in the paper. Recall that $\mathcal{C} = \mathcal{C}(\mathcal{I})$.

▶ **Lemma 12.** *Let $\mathcal{I} = (I, w, p, m)$ be a dVMK instance, and $h : I \to \mathbb{R}_{\geq 0}$. Define $f : \mathcal{C}^\ell \to \mathbb{R}$ by $f(C_1, \ldots, C_\ell) = \frac{h(\bigcup_{i \in [\ell]} C_i)}{\eta}$, where $\eta \geq \max_{C \in \mathcal{C}} h(C)$. Then f is a self-bounding function.*

3.2 Profit of the Sampled Configurations

In this section we prove Lemma 7; namely, we show that with high probability the profit $p(\cup_{t \in \{1,\ldots,\ell\}} R_t)$ is sufficiently large. We first prove the next lemma.

▶ **Lemma 13.** $\mathbb{E}[p(T)] \geq (1 - e^{-\alpha} - \varepsilon) \cdot p(\mathrm{OPT})$.

Proof. Let $\mathcal{C}(i) = \mathcal{C}(\mathcal{I}, i)$ be the set of configurations containing item $i \in \mathcal{I}$, then for every $i \in I$, the probability that i is not contained in the sampled configurations is

$$\Pr[i \notin T] = \Pr[i \notin \cup_{t \in \{1,\ldots,\ell\}} R_t] = \prod_{t \in \{1,\ldots,\ell\}} \Pr[i \notin R_t] = \left(1 - \sum_{C \in \mathcal{C}(i)} \frac{x_C}{m}\right)^{\ell},$$

where third equality holds since $\Pr[i \in R_t] = \sum_{C \in \mathcal{C}(i)} \frac{x_C}{m}$, for all $t \in \{1, \ldots, \ell\}$. Hence,

$$\Pr[i \notin T] = \left(1 - \sum_{C \in \mathcal{C}(i)} \frac{x_C}{m}\right)^{\ell} \leq \left(1 - \sum_{C \in \mathcal{C}(i)} \frac{x_C}{m}\right)^{m \cdot \alpha \cdot \frac{\sum_{C \in \mathcal{C}(i)} x_C}{\sum_{C \in \mathcal{C}(i)} x_C}} \leq \exp\left(-\alpha \cdot \sum_{C \in \mathcal{C}(i)} x_C\right).$$

The first inequality holds since $\ell \geq \alpha \cdot m$. The second inequality holds by $(1-\frac{1}{x})^x \leq e^{-1}$ for $x \geq 1$. Thus, we have

$$\Pr[i \in T] = 1 - \Pr[i \notin T] \quad \geq \left(1 - \exp\left(-\alpha \cdot \sum_{C \in \mathcal{C}(i)} x_C\right)\right) \geq \sum_{C \in \mathcal{C}(i)} x_C \cdot (1 - e^{-\alpha})$$

For the second inequality, we used $1 - e^{-x \cdot \alpha} \geq x \cdot (1 - e^{-\alpha})$ for $x, \alpha \in [0,1]$. Therefore,

$$\begin{aligned}
\mathbb{E}[p(T)] &= \sum_{i \in I} p(i) \cdot \Pr[i \in T] \\
&\geq \sum_{i \in I} p(i) \cdot \sum_{C \in \mathcal{C}(i)} x_C \cdot (1 - e^{-\alpha}) \\
&= (1 - e^{-\alpha}) \cdot \sum_{C \in \mathcal{C}} \sum_{i \in C} x_C \cdot p(i) \\
&= (1 - e^{-\alpha}) \cdot x^* \\
&\geq (1 - e^{-\alpha} - \varepsilon) \cdot p(\mathrm{OPT}).
\end{aligned}$$

The third equality follows from our definition of x^* as the value of the solution $\mathbf{x}$ found in Step 1 of Algorithm 1. The second inequality holds since $x^* \geq (1-\varepsilon) \cdot p(\mathrm{OPT})$. ◀

Proof of Lemma 7. Define $f : C^\ell \to \mathbb{R}$ by $f(C_1, \ldots, C_\ell) = \frac{p(\bigcup_{t \in \{1,\ldots,\ell\}} C_t)}{\varepsilon^{10} \cdot p(\mathrm{OPT})}$ As the instance $\mathcal{I}$ is ε-nice, we have that $\varepsilon^{10} \cdot p(\mathrm{OPT}) \geq \max_{C \in \mathcal{C}} p(C)$. By Lemma 12, f is a self-bounding function. Hence, by Lemma 13 we have,

$$\begin{aligned}
&\Pr\left[p(T) \leq (1 - e^{-\alpha} - 2 \cdot \varepsilon) \cdot p(\mathrm{OPT})\right] \\
&\quad \leq \Pr\left[\frac{p(T)}{\epsilon^{10} \cdot p(OPT)} \leq \frac{\mathbb{E}[p(T)]}{\epsilon^{10} \cdot p(\mathrm{OPT})} - \frac{\varepsilon \cdot p(\mathrm{OPT})}{\epsilon^{10} \cdot p(\mathrm{OPT})}\right] \\
&\quad = \Pr\left[f(R_1, \ldots, R_\ell) \leq \mathbb{E}[f(R_1, \ldots, R_\ell)] - \varepsilon^{-9}\right] \\
&\quad \leq \exp\left(-\frac{\varepsilon^{-18}}{2 \cdot \mathbb{E}[f(R_1, \ldots, R_\ell)]}\right) \\
&\quad \leq \exp\left(-\frac{\varepsilon^{-18}}{2 \cdot \frac{p(\mathrm{OPT})}{\epsilon^{10} \cdot p(\mathrm{OPT})}}\right) \leq \exp\left(-\varepsilon^{-7}\right).
\end{aligned}$$

The first equality holds by the definition of f. The second inequality follows from Lemma 11, by taking $t = \varepsilon^{-9}$. The third inequality holds since $f(R_1, \ldots, R_\ell) \leq \frac{p(\mathrm{OPT})}{\varepsilon^{10} \cdot p(\mathrm{OPT})}$, as $R_1, \ldots, R_\ell$ along with additional $m - \ell$ empty configurations is a solution for $\mathcal{I}$. The fourth inequality holds since $2 \cdot \varepsilon \leq 1$. ◀

3.3 The Solution for the Residual Items

In this section we prove Lemma 8. Specifically, we show that the profit of the solution for the MK instance constructed in Step 4 of Algorithm 1 is sufficiently high. Since we obtain a $(1-\varepsilon)$-approximate solution for the MK instance $\mathcal{I}'$, we only need to derive a lower bound for $\mathrm{OPT}(\mathcal{I}')$. To this end, we show that there exists a set $Q \subseteq \mathrm{OPT}$, such that the set $Q \setminus T$ has sufficiently high profit $p(Q \setminus T)$, and $Q \setminus T$ can be almost entirely packed in twice the number of remaining bins. We choose among these bins the most profitable ones to obtain the lower bound. In our analysis, we use the notion of subset-obliviousness, introduced in [2]. The following is a simplified version of a definition given in [2] w.r.t. the BIN PACKING (BP) problem. Let $\texttt{BP-OPT}(I, w)$ denote the size of an optimal solution for a BP instance $\mathcal{I} = (I, w)$.

▶ **Definition 14.** *Let $\rho > 1$. We say that* Bin Packing *is ρ-subset oblivious if, for any fixed $\varepsilon > 0$, there exist k, ψ, δ (possibly depending on ε) such that, for any BP instance $\mathcal{I} = (I, w)$, there exist functions $g_1, \ldots, g_k : 2^I \to \mathbb{R}_{\geq 0}$ which satisfy the following.*

(i) *$g_t(C) \leq \psi$ for any $C \in \mathcal{C}(\mathcal{I})$ and $t \in \{1, \ldots, k\}$;*

(ii) $\texttt{BP-OPT}(I, w) \geq \max_{t \in \{1,\ldots,k\}} g_t(I)$*;*

(iii) $\texttt{BP-OPT}(S, w) \leq \rho \cdot \max_{t \in \{1,\ldots,k\}} g_t(S) + \varepsilon \cdot \texttt{BP-OPT}(I, w) + \delta$*, for all $S \subseteq I$.*

We refer to the values k, ψ and δ as the (ρ, ε)-subset oblivious parameters of Bin Packing*, and the functions $g_1, \ldots, g_k$ as the (ρ, ε)-subset oblivious functions of $\mathcal{I}$.*

The next lemma follows from a result of [2].

▶ **Lemma 15.** *For any fixed $\varepsilon > 0$,* Bin Packing *is $(1+\varepsilon)$-subset oblivious, and the $(1+\varepsilon, \varepsilon)$ parameters k, δ, ψ satisfy $k \leq \exp^{[3]}(\varepsilon^{-1})$, $\delta \leq \frac{4}{\varepsilon^4}$, and $\psi \leq 1$.*

Let $\mathcal{J} = (\mathrm{OPT}, w', p, 2 \cdot m)$ be the 2-associated MK instance of $\mathcal{I}_{\mathrm{OPT}} = (\mathrm{OPT}, w, p, m)$. By Lemma 15, Bin Packing is $(1+\varepsilon^2)$-subset oblivious. Thus, there exist k, ψ, δ which are $(1+\varepsilon^2, \varepsilon^2)$ subset oblivious parameters of Bin Packing. Let $g_1, \ldots, g_k$ be the $(1+\varepsilon^2, \varepsilon^2)$ subset-oblivious functions of the Bin Packing instance (OPT, w'). By Lemma 15, the values k, ψ and δ satisfy, $k \leq \exp^{[3]}(\varepsilon^{-2})$, $\delta \leq \frac{4}{\varepsilon^8}$, $\psi \leq 1$. Define $\alpha' = \frac{\lceil \alpha \cdot m \rceil}{m}$, then as $\alpha' \cdot m - \alpha \cdot m \leq 1$, we have that $\alpha' - \alpha \leq \frac{1}{m} \leq \varepsilon$.

▶ **Lemma 16.** *There exists $Q \subseteq \mathrm{OPT}$ which satisfies the following.*

1. $\Pr\left[g_t(Q \setminus T) \geq (1-\alpha') \cdot g_t(\mathrm{OPT}) + k \cdot \psi + \varepsilon^{10} \cdot m\right] \leq \exp\left(-\frac{\varepsilon^{21} \cdot m}{\psi^2}\right)$*, for all $t \in \{1, \ldots, k\}$.*
2. $\Pr\left[p(Q \setminus T) \leq (1 - \alpha' - \varepsilon) \cdot p(\mathrm{OPT})\right] \leq \exp\left(-\varepsilon^{-7}\right)$.

Proof. To show the existence of the set Q satisfying the properties in the lemma, consider first the following optimization problem. Given an optimal solution OPT for a 2VMK instance $\mathcal{I}$, find a subset of items $Q \in \mathrm{OPT}$ for which $\mathbb{E}\left[p(Q \setminus T)\right]$ is maximized, under the constraint that $\mathbb{E}\left[g_t(Q \setminus T)\right] \leq (1-\alpha')g_t(\mathrm{OPT})$ for all $t \in [k]$. Let $y_i \in \{0, 1\}$ be an indicator for the inclusion of item $i \in \mathrm{OPT}$ in Q. We can formulate an integer program for the above optimization problem. In the following LP relaxation we have $0 \leq y_i \leq 1$, $\forall\ i \in \mathrm{OPT}$.

$$
\begin{aligned}
(Q\text{-LP}) \quad \max \quad & \sum_{i \in \mathrm{OPT}} y_i \cdot p(i) \cdot \Pr[i \notin T] \\
\text{s.t.} \quad & \sum_{i \in \mathrm{OPT}} y_i \cdot \Pr[i \notin T] \cdot g_t(i) \leq (1-\alpha') g_t(\mathrm{OPT}) \quad \forall t \in \{1, \ldots, k\} \qquad (5) \\
& 0 \leq y_i \leq 1 \qquad \forall i \in \mathrm{OPT}
\end{aligned}
$$

Let $\mathbf{y}^*$ be a basic optimal solution for Q-LP. We define

$$Q = \{i \in Q \mid y_i^* > 0\} \qquad (6)$$

to be the set of all items with positive entries in $\mathbf{y}^*$. We show that the set Q defined in (6) satisfies $\mathbb{E}\left[p(Q \setminus T)\right] \geq (1-\alpha')p(\mathrm{OPT})$. To this end, we prove the next claim.

▷ Claim 17. $\sum_{i \in \mathrm{OPT}} y_i^* \cdot p(i) \cdot \Pr[i \notin T] \geq (1-\alpha') \cdot p(\mathrm{OPT})$.

Proof. For every $i \in \mathrm{OPT}$, the following holds:

$$
\begin{aligned}
\Pr[i \in T] &= \Pr\left[\exists t \in \{1, \ldots, \ell\}, i \in R_t\right] \\
&\leq \sum_{t \in \{1,\ldots,\ell\}} \Pr\left[i \in R_t\right] \\
&= \sum_{t \in \{1,\ldots,\ell\}} \sum_{C \in \mathcal{C}(i)} \frac{x_C}{m} \leq \sum_{t \in \{1,\ldots,\ell\}} \frac{1}{m} = \frac{\ell}{m}.
\end{aligned}
$$

Consider the vector $\mathbf{y}' = (y'_1, \ldots, y'_{|\mathrm{OPT}|})$ where $y'_i = \dfrac{1-\alpha'}{\Pr[i \notin T]}$ for every $i \in \mathrm{OPT}$. Then $\mathbf{y}'$ is a feasible solution for Q-LP since the following holds:

1. For every $i \in \mathrm{OPT}$, $y'_i = \frac{1-\alpha'}{\Pr[i \notin T]}$ satisfies the following,

$$0 \leq \frac{1-\alpha'}{\Pr[i \notin T]} = \frac{1-\alpha'}{1-\Pr[i \in T]} \leq \frac{1-\alpha'}{1-\frac{\ell}{m}} = \frac{1-\alpha'}{1-\frac{\alpha' \cdot m}{m}} = 1.$$

   Therefore $0 \leq y'_i \leq 1$, for all $i \in \mathrm{OPT}$.
2. For every $t \in \{1, \ldots, k\}$, the following holds:

$$\sum_{i \in \mathrm{OPT}} y'_i \cdot \Pr[i \notin T] \cdot g_t(i) = (1-\alpha') \cdot \sum_{i \in \mathrm{OPT}} g_t(i) = (1-\alpha') \cdot g_t(\mathrm{OPT}).$$

The objective value $\sum_{i \in \mathrm{OPT}} y'_i \cdot p(i) \cdot \Pr[i \notin T]$ satisfies:

$$\sum_{i \in \mathrm{OPT}} y'_i \cdot p(i) \cdot \Pr[i \notin T] = \sum_{i \in \mathrm{OPT}} (1-\alpha') \cdot p(i) = (1-\alpha') \cdot p(\mathrm{OPT}).$$

This implies that the objective value of an optimal solution for Q-LP is at least $(1-\alpha') \cdot p(\mathrm{OPT})$. Hence, $\sum_{i \in \mathrm{OPT}} y^*_i \cdot p(i) \cdot \Pr[i \notin T] \geq (1-\alpha') \cdot p(\mathrm{OPT})$. ◁

▷ Claim 18. The subset Q satisfies the following properties.

1. $\mathbb{E}[g_t(Q \setminus T)] \leq (1-\alpha') g_t(\mathrm{OPT}) + k \cdot \psi$, for every $t \in \{1, \ldots, k\}$.
2. $\mathbb{E}[p(Q \setminus T)] \geq (1-\alpha') p(\mathrm{OPT})$.

Proof. The basic optimal solution $\mathbf{y}^*$ has at least $|\mathrm{OPT}|$ tight constraints. Therefore, at least $|\mathrm{OPT}| - k$ constraints of the form $y_i \geq 0$ or $y_i \leq 1$ are tight, i.e., we have at least $|\mathrm{OPT}| - k$ variables y_i with tight constraint. Let $B = \{i \in \mathrm{OPT} \mid 0 < y^*_i < 1\}$ the set of fractional variables, then $|B| \leq k$. For every $t \in \{1, \ldots, k\}$, the following holds:

$$\begin{aligned}
\mathbb{E}[g_t(Q \setminus T)] &= \sum_{i \in Q} 1 \cdot \Pr[i \notin T] \cdot g_t(i) \\
&= \sum_{i \in B} \Pr[i \notin T] \cdot g_t(i) + \sum_{i \in Q \setminus B} y^*_i \cdot \Pr[i \notin T] \cdot g_t(i) \\
&\leq k \cdot \psi + \sum_{i \in Q \setminus B} y^*_i \cdot \Pr[i \notin T] \cdot g_t(i) \\
&\leq (1-\alpha') \cdot g_t(\mathrm{OPT}) + k \cdot \psi.
\end{aligned}$$

The second equality holds since $y^*_i = 1$, for every $i \in Q \setminus B$. The first inequality holds since $C = \{i\} \in \mathcal{C}(\mathcal{J})$ is a configuration, for every $i \in B$. Therefore, $g_t(C) \leq \psi$, and $|B| \leq k$. The second inequality follows from the constraints of Q-LP. Furthermore,

$$\mathbb{E}[p(Q \setminus T)] = \sum_{i \in Q} 1 \cdot p(i) \cdot \Pr[i \notin T] \quad \geq \sum_{i \in Q} y^*_i \cdot p(i) \cdot \Pr[i \notin T] \geq (1-\alpha') \cdot p(\mathrm{OPT}).$$

The first inequality holds since $y^*_i \leq 1$, for every $i \in Q$. The second inequality follows from Claim 17. ◁

We now show that the set Q defined in (6) satisfies properties 1. and 2. in the lemma.

▷ Claim 19. For every $t \in \{1, \ldots, k\}$,

$$\Pr\left[g_t(Q \setminus T) \geq (1-\alpha') \cdot g_t(\mathrm{OPT}) + k \cdot \psi + \varepsilon^{10} \cdot m\right] \leq \exp\left(-\frac{\varepsilon^{21} \cdot m}{\psi^2}\right).$$

Proof. Let $t \in \{1, \ldots, k\}$. Define $q : \mathcal{C}(\mathcal{I}) \to \mathcal{C}(\mathcal{I}_{\mathrm{OPT}})$ by $q(C) = C \cap Q$ for every $C \in \mathcal{C}(\mathcal{I})$. Also, define $f : \mathcal{C}^\ell \to \mathbb{R}$ by

$$f(C_1, \ldots, C_\ell) = \frac{g_t\left(\bigcup_{r\in\{1,\ldots,\ell\}} q(C_r)\right)}{2 \cdot \psi}$$

for every $(C_1, \ldots, C_\ell) \in \mathcal{C}^\ell$, where $\mathcal{C} = \mathcal{C}(\mathcal{I})$. Since $q(C_r) \in \mathcal{C}(\mathcal{I}_{\mathrm{OPT}})$, for every $r \in \{1, \ldots, \ell\}$, by Lemma 4, there exists $C_1, C_2 \in \mathcal{J}$, such that $C_1 \cup C_2 = q(C_r)$. Thus, $2 \cdot \psi \geq g_t(C_1) + g_t(C_2) \geq g_t(q(C))$. By Lemma 12, f is self-bounding function. We note that

$$\mathbb{E}\left[g_t(T \cap Q)\right] \leq \mathbb{E}\left[g_t(\mathrm{OPT})\right] \leq 2 \cdot m \cdot \psi.$$

The first inequality holds since $T \cap Q \subseteq Q \subseteq \mathrm{OPT}$. By Lemma 4, there exist $2 \cdot m$ configurations in $\mathcal{C}(\mathcal{J})$, whose union is OPT, and each configuration $C \in \mathcal{C}(\mathcal{J})$ satisfies $g_t(C) \leq \psi$; thus, the second inequality holds. Hence, we have

$$\begin{aligned}
&\Pr\left[g_t(Q \setminus T) \geq (1-\alpha') \cdot g_t(\mathrm{OPT}) + k \cdot \psi + \varepsilon^{10} \cdot m\right] \\
&\quad \leq \Pr\left[g_t(Q \setminus T) \geq \mathbb{E}\left[g_t(Q \setminus T)\right] + \varepsilon^{10} \cdot m\right] \\
&\quad = \Pr\left[\frac{g_t(T \cap Q)}{2 \cdot \psi} \leq \mathbb{E}\left[\frac{g_t(T \cap Q)}{2 \cdot \psi}\right] - \frac{\varepsilon^{10} \cdot m}{2 \cdot \psi}\right] \\
&\quad = \Pr\left[f(R_1, \ldots, R_\ell) \leq \mathbb{E}\left[f(R_1, \ldots, R_\ell)\right] - \frac{\varepsilon^{10} \cdot m}{2 \cdot \psi}\right].
\end{aligned}$$

The first inequality holds by Claim 18. The first equality holds since $g(Q \setminus T) = g(Q) - g(Q \cap T)$. Thus,

$$\begin{aligned}
&\Pr\left[g_t(Q \setminus T) \geq (1-\alpha') \cdot g_t(\mathrm{OPT}) + k \cdot \psi + \varepsilon^{10} \cdot m\right] \\
&\quad \leq \Pr\left[f(R_1, \ldots, R_\ell) \leq \mathbb{E}\left[f(R_1, \ldots, R_\ell)\right] - \frac{\varepsilon^{10} \cdot m}{2 \cdot \psi}\right] \\
&\quad \leq \exp\left(-\frac{\left(\frac{\varepsilon^{10} \cdot m}{2 \cdot \psi}\right)^2}{2 \cdot \mathbb{E}[f(R_1, \ldots, R_\ell)]}\right) \leq \exp\left(-\frac{\varepsilon^{20} \cdot m^2}{2 \cdot 2m \cdot 4 \cdot \psi^2}\right) \leq \exp\left(-\frac{\varepsilon^{21} \cdot m}{\psi^2}\right).
\end{aligned}$$

For the first inequality we used Lemma 11 with $t = \frac{\varepsilon^{10} \cdot m}{2 \cdot \psi}$. The second inequality holds since $\mathbb{E}\left[f(R_1, \ldots, R_\ell)\right] = \frac{\mathbb{E}[g_t(T \cap Q)]}{2 \cdot \psi} \leq 2 \cdot m$. The third inequality holds since $\varepsilon \cdot 16 \leq 1$. ◁

▷ Claim 20. $\Pr\left[p(Q \setminus T) \leq (1 - \alpha' - \varepsilon) \cdot p(\mathrm{OPT})\right] \leq \exp\left(-\varepsilon^{-7}\right)$.

Proof of Claim 20. Let $\tilde{p} : I \to \mathbb{R}_{\geq 0}$ such that $\tilde{p}(i) = p(i)$ for $i \in Q$, and $\tilde{p}(i) = 0$ for $i \notin Q$. Define $f : \mathcal{C}^\ell \to \mathbb{R}$ by

$$f(C_1, \ldots, C_\ell) = \frac{\tilde{p}(\bigcup_{i\in[\ell]} C_i)}{\varepsilon^{10} \cdot p(\mathrm{OPT})}$$

for every $(C_1, \ldots, C_\ell) \in \mathcal{C}^\ell$, where $\mathcal{C} = \mathcal{C}(\mathcal{I})$. Since the instance $\mathcal{I}$ is ε-*nice*,

$$\varepsilon^{10} \cdot p(\mathrm{OPT}) \geq \max_{C\in\mathcal{C}(\mathcal{I})} p(C) \geq \max_{C\in\mathcal{C}(\mathcal{I})} \tilde{p}(C).$$

Therefore, by Lemma 12, f is self bounding function. Now, we can use Lemma 11 and get the following.

$$\begin{aligned}
&\Pr\left[p(Q\setminus T)\le(1-\alpha')\cdot p(\mathrm{OPT})-\varepsilon\cdot p(\mathrm{OPT})\right]\\
&\quad\le\Pr\left[p(Q\setminus T)\le\mathbb{E}\left[p(Q\setminus T)\right]-\varepsilon\cdot p(\mathrm{OPT})\right]\\
&\quad=\Pr\left[\tilde{p}(T)\ge\mathbb{E}\left[\tilde{p}(T)\right]+\varepsilon\cdot p(\mathrm{OPT})\right]\\
&\quad=\Pr\left[\frac{\tilde{p}(T)}{\varepsilon^{10}\cdot p(\mathrm{OPT})}\ge\mathbb{E}\left[\frac{\tilde{p}(T)}{\varepsilon^{10}\cdot p(\mathrm{OPT})}\right]+\varepsilon^{-9}\right]\\
&\quad=\Pr\left[f(R_1,\ldots,R_\ell)\ge\mathbb{E}\left[f(R_1,\ldots,R_\ell)\right]+\varepsilon^{-9}\right].
\end{aligned}$$

The first inequality holds by Claim 18. The first equality follows from subtracting $p(Q)$ for both sides and using $p(Q\setminus T)=p(Q)-p(Q\cap T)$. Thus,

$$\begin{aligned}
&\Pr\left[p(Q\setminus T)\le(1-\alpha')\cdot p(\mathrm{OPT})-\varepsilon\cdot p(\mathrm{OPT})\right]\\
&\quad\le\Pr\left[f(R_1,\ldots,R_\ell)\ge\mathbb{E}\left[f(R_1,\ldots,R_\ell)\right]+\varepsilon^{-9}\right]\\
&\quad\le\exp\left(-\frac{\varepsilon^{-18}}{2\cdot\mathbb{E}[f(R_1,\ldots,R_\ell)]+\frac{\varepsilon^{-9}}{3}}\right)\\
&\quad\le\exp\left(-\frac{\varepsilon^{-18}}{2\cdot\frac{p(\mathrm{OPT})}{\varepsilon^{10}\cdot p(\mathrm{OPT})}+\frac{\varepsilon^{-9}}{3}}\right)\\
&\quad\le\exp\left(-\varepsilon^{-7}\right).
\end{aligned}$$

The first inequality follows from using Lemma 11 with $t=\varepsilon^{-9}$. In the second inequality we used the inequality $\mathbb{E}\left[\frac{\tilde{p}(T)}{\varepsilon^{10}\cdot p(\mathrm{OPT})}\right]\le\frac{p(\mathrm{OPT})}{\varepsilon^{10}\cdot p(\mathrm{OPT})}$. The third inequality holds since $\varepsilon^{-10}\ge\frac{\varepsilon^{-9}}{3}$ and $\varepsilon\cdot 3\le 1$. ◁

By Claim 19 and Claim 20, we have the statement of the lemma. ◀

Proof of Lemma 8. Let Q be the set defined in Lemma 16, and let F_t be the event "$g_t(Q\setminus T)\le(1-\alpha')\cdot g_t(S)+k\cdot\psi+\varepsilon^{10}\cdot m$", for every $t\in\{1,\ldots,k\}$. Also, let F_p be the event "$p(Q\setminus T)\ge(1-\alpha'-\varepsilon)\cdot p(\mathrm{OPT})$". By Lemma 16, the following holds:

1. $\Pr[F_t]\ge 1-\exp(-\frac{\varepsilon^{21}\cdot m}{\psi^2})$, for every $t\in\{1,\ldots,k\}$.
2. $\Pr[F_p]\ge 1-\exp(-\varepsilon^{-7})$.

Therefore, by the union bound, we have

$$\begin{aligned}
\Pr\left[F_p\cap\bigcap_{t\in\{0,\ldots,k\}}F_t\right]&\ge 1-\left(\exp(-\varepsilon^{-7})+\sum_{t\in\{0,\ldots,k\}}\exp\left(-\frac{\varepsilon^{21}\cdot m}{\psi^2}\right)\right)\\
&=1-\exp(-\varepsilon^{-7})-k\cdot\exp\left(-\frac{\varepsilon^{21}\cdot m}{\psi^2}\right)\ge\frac{3}{4}.
\end{aligned}$$

The second inequality holds since $k\le\exp^{[3]}(\varepsilon^{-2})$, $\psi\le 1$, and $m\ge\exp^{[3]}(\varepsilon^{-30})$.

▷ Claim 21. Assuming that $F_p\cap\bigcap_{t\in\{1,\ldots,k\}}F_t$ occurs,

$$\texttt{BP-OPT}(Q\setminus T,w')\le\left(2\cdot(1-\alpha')+6\cdot\varepsilon^2\right)\cdot m.$$

Proof. We note that

$$\begin{aligned}
\max_{t\in\{1,\ldots,k\}}g_t(Q\setminus T)&\le(1-\alpha')\cdot\max_{t\in\{1,\ldots,k\}}\{g_t(\mathrm{OPT})\}+k\cdot\psi+\varepsilon^{10}\cdot m\\
&\le(1-\alpha')\cdot\texttt{BP-OPT}(\mathrm{OPT},w')+k\cdot\psi+\varepsilon^{10}\cdot m\\
&\le\left(2\cdot(1-\alpha')+\varepsilon^{10}\right)\cdot m+k\cdot\psi.
\end{aligned}$$

The first inequality holds since $\bigcap_{t\in\{1,\ldots,k\}} F_t$ occurs. The second inequality holds since $g_1,\ldots,g_k$ are the $(1+\varepsilon^2,\varepsilon^2)$-subset oblivious functions of (OPT,w'). The third inequality holds since there exist m configurations in $\mathcal{C}(\mathcal{I})$ whose union is OPT. Therefore, by Lemma 4, there exist in $\mathcal{C}(\mathcal{J})$ $2\cdot m$ configurations whose union is OPT. Thus, $\texttt{BP-OPT}(\mathrm{OPT},w')\leq 2\cdot m$. We have that

$$\begin{aligned}
\texttt{BP-OPT}(Q\setminus T,w') &\leq (1+\varepsilon^2)\cdot \max_{t\in\{1,\ldots,k\}}\{g_t(Q\setminus T)\}+\varepsilon^2\cdot\texttt{BP-OPT}(\mathrm{OPT},w')+\delta\\
&\leq (1+\varepsilon^2)\cdot\left(\left(2\cdot(1-\alpha')+\varepsilon^{10}\right)\cdot m+k\cdot\psi\right)+\varepsilon^2\cdot 2\cdot m+\delta\\
&\leq \left(2\cdot(1-\alpha')+5\cdot\varepsilon^2\right)\cdot m+2\cdot k\cdot\psi+\delta\\
&\leq \left(2\cdot(1-\alpha')+5\cdot\varepsilon^2\right)\cdot m+2\cdot\exp^{[3]}(\varepsilon^{-2})+\frac{4}{\varepsilon^8}\\
&\leq \left(2\cdot(1-\alpha')+6\cdot\varepsilon^2\right)\cdot m.
\end{aligned}$$

The first inequality follows from Definition 14. The second inequality follows from Lemma 4. The fourth inequality holds since $k\leq\exp^{[3]}(\varepsilon^{-2})$ and $\delta\leq\frac{4}{\varepsilon^8}$. The fifth inequality holds since $\varepsilon^2\cdot m\geq 2\cdot\exp^{[3]}(\varepsilon^{-2})+\frac{4}{\varepsilon^8}$. ◁

By Claim 21, there exist $\left(2\cdot(1-\alpha')+6\cdot\varepsilon^2\right)\cdot m$ configurations in $\mathcal{C}(\mathcal{J})$ whose union is $Q\setminus T$. Among these configurations, we choose the $(1-\alpha')\cdot m$ most profitable. Let R be the items chosen in these configurations. Then,

$$\begin{aligned}
p(R) &\geq \frac{(1-\alpha')\cdot m}{(2\cdot(1-\alpha')+6\cdot\varepsilon^2)\cdot m}\cdot p(Q\setminus T)\\
&\geq \frac{(1-\alpha')\cdot m}{(2\cdot(1-\alpha')+6\cdot\varepsilon^2)\cdot m}\cdot(1-\alpha'-\varepsilon)\cdot p(\mathrm{OPT})\\
&= \frac{1-\alpha'}{2\cdot(1-\alpha')+6\cdot\varepsilon^2}\cdot(1-\alpha'-\varepsilon)\cdot p(\mathrm{OPT})\\
&\geq \frac{1-\alpha'-\varepsilon}{2+\varepsilon}\cdot p(\mathrm{OPT})\geq\left(\frac{1-\alpha-2\varepsilon}{2}-\frac{\varepsilon}{4}\right)\cdot p(\mathrm{OPT})\geq\left(\frac{1-\alpha}{2}-2\cdot\varepsilon\right)\cdot p(\mathrm{OPT}).
\end{aligned}$$

The third inequality holds since $\frac{6\cdot\varepsilon^2}{1-\alpha'}\leq\varepsilon$. The fourth inequality follows from

$$\frac{1-\alpha'-\varepsilon}{2+\varepsilon}\geq\frac{1-\alpha'-\varepsilon}{2}-\frac{(1-\alpha'-\varepsilon)\cdot\varepsilon}{4}\geq\frac{1-\alpha'-\varepsilon}{2}-\frac{\varepsilon}{4},$$

and by using $\alpha'\leq\alpha+\varepsilon$. Therefore, with probability at least $\frac{3}{4}$, the optimal profit of the MK instance $(Q\setminus T,w',p,(1-\alpha')\cdot m)$ is at least $p(R)$. Since $Q\setminus T\subseteq I\setminus T$, the optimal profit of the MK instance $(I\setminus T,w',p,(1-\alpha')\cdot m)$ is at least $p(R)$. As we obtain a $1-\varepsilon$-approximation for $\mathcal{I}'$ (using, e.g., [10]), the profit of Step 4 in Algorithm 1 is at least

$$(1-\varepsilon)\cdot p(R)\geq\left(\frac{1-\alpha}{2}-2\cdot\varepsilon\right)\cdot(1-\varepsilon)\cdot p(\mathrm{OPT})\quad\geq\left(\frac{1-\alpha}{2}-3\cdot\varepsilon\right)\cdot p(\mathrm{OPT}).$$ ◀

4 APX-hardness

In this section we prove Theorem 1. We use a reduction from 2VBP to show that a PTAS for 2VMK would imply the existence of an APTAS for 2VBP. We rely on the following result of Ray [17], which addresses a flaw in an earlier proof of Woeginger [21].

▶ **Theorem 22** ([17]). *Assuming* P $\neq$ NP*, there is no* APTAS *for* 2VBP*.*

Proof of Theorem 1. Assume towards a contradiction that there is a PTAS $\{\mathcal{A}_\varepsilon\}$ for 2VMK. That is, for every $\varepsilon > 0$, $\mathcal{A}_\varepsilon$ is a polynomial-time $(1-\varepsilon)$-approximation algorithm for 2VMK.

In Algorithm 2 we use $\{A_\varepsilon\}$ to derive an APTAS for 2VBP. The algorithm calls as a subroutine algorithm First-Fit (FF). The input for FF is an instance $\mathcal{I}$ of 2VMK. The output is a feasible packing of all items in $\mathcal{I}$ in a set of 2-dimensional bins with unit size in each dimension. First-Fit proceeds by considering the items in arbitrary order and assigning the next item in the list to the first bin which can accommodate the item. If no such bin exists, FF opens a new bin and assigns the item to this bin. (For more details on FF and its analysis see, e.g., [19].) Given a 2VBP instance, we use in Algorithm 2 the notion of associated 2VMK instance.

▶ **Definition 23.** *Given a* 2VBP *instance* $\mathcal{I} = (I, w)$ *and* $t \geq 1$*, we define its* t*-associated* 2VMK *instance* $\mathcal{I}' = (I', w', p', m')$ *as follows. The set of items is* $I' = I$*, and the weight function is* $w' = w$*. The profit of each item* $i \in I'$ *is* $p'(i) = w_1(i) + w_2(i)$*, and the number of bins is* $m' = t$*.*

Algorithm 2 Reduction from 2VBP.

configuration : A PTAS $\{\mathcal{A}_\varepsilon\}$ for 2VMK and $\varepsilon > 0$
input : A 2VBP instance $\mathcal{I} = (I, w)$.
output : A solution for $\mathcal{I}$ which uses at most $(1+\varepsilon) \cdot \texttt{BP-OPT}(\mathcal{I}) + 2$ bins.

1 **for** $t = 1$ *to* $|I|$ **do**
2 Let $\mathcal{I}' = (I', w', p', m')$ be the t-associated 2VMK instance of $\mathcal{I}$
3 Use $\mathcal{A}_{\varepsilon'}$ to solve $\mathcal{I}'$, where $\varepsilon' = \frac{\varepsilon}{16}$. Let $C_1, \ldots, C_{m'} \subseteq I$ be the returned solution.
4 Define $S = \bigcup_{j=1}^{m'} C_j$ and pack the residual items $I \setminus S$ using First-Fit.
5 Add $C_1, \ldots, C_m$ along with the packing of S to a list of candidate solutions.
6 **end**
7 Return the candidate solution with the smallest number of bins used.

▷ Claim 24. For any 2VBP instance $\mathcal{I}$ and $\varepsilon > 0$, Algorithm 2 returns a solution which uses at most $(1+\varepsilon) \cdot \texttt{BP-OPT}(\mathcal{I}) + 2$ bins.

Proof. Let $R = \texttt{BP-OPT}(\mathcal{I})$ and let $\mathcal{I}' = (I', w', p', m')$ be the R-associated instance of $\mathcal{I}$. Note that $\mathrm{OPT}(\mathcal{I}') = \sum_{i \in I} p'(i)$, since we can pack all the items I in R bins with the weight function w. Consider the iteration of Step 1 in Algorithm 2 where $t = R$. We show that the number of bins used by the solution is at most $(1+\varepsilon) \cdot \texttt{BP-OPT}(I, w) + 2$. Observe that S, the set in Step 4 of Algorithm 2, satisfies $p'(S) \geq (1 - \frac{\varepsilon}{16}) \cdot \mathrm{OPT}(\mathcal{I}')$. Thus, $p'(I \setminus S) \leq \frac{\varepsilon}{16} \cdot \mathrm{OPT}(\mathcal{I}')$. It follows that

$$\begin{aligned} w_1(I \setminus S) + w_2(I \setminus S) &\leq \frac{\varepsilon}{16} \cdot \mathrm{OPT}(\mathcal{I}') \\ &= \frac{\varepsilon}{16} \cdot (w_1(I) + w_2(I)) \\ &\leq \frac{\varepsilon}{16} \cdot 2\max\{w_1(I), w_2(I)\} \\ &\leq \frac{\varepsilon \cdot \texttt{BP-OPT}(I, w)}{8}. \end{aligned}$$

The third inequality holds since every bin $b \in \mathcal{C}(\mathcal{I}')$ satisfies $w_1(b), w_2(b) \leq 1$. In the packing of the residual items $I \setminus S$ using First-Fit, every two consecutive bins b_1, b_2 satisfy

$w_1(b_1 \cup b_2) + w_2(b_1 \cup b_2) > 1$. Assume the items in $I \setminus S$ are packed in at least $\varepsilon \cdot \texttt{BP-OPT}(I, w) + 2$ bins in Step 4 of Algorithm 2. Then,

$$w_1(I \setminus S) + w_2(I \setminus S) \geq \left\lfloor \frac{\varepsilon \cdot \texttt{BP-OPT}(I, w) + 2}{2} \right\rfloor > \frac{\varepsilon \cdot \texttt{BP-OPT}(I, w) + 2}{2} - 1 \geq w_1(I \setminus S) + w_2(I \setminus S)$$

The first inequality holds since there are at least $\left\lfloor \frac{\varepsilon \cdot \texttt{BP-OPT}(I,w)+2}{2} \right\rfloor$ disjoint pairs of consecutive bins, and each pair b_1, b_2 satisfies $w_1(b_1 \cup b_2) + w_2(b_1 \cup b_2) \geq 1$. This is a contradiction. Therefore, at most $\varepsilon \cdot \texttt{BP-OPT}(I, w) + 2$ bins are used for the items $I \setminus S$. Hence, Algorithm 2 returns a solution with at most $(1 + \varepsilon) \cdot \texttt{BP-OPT}(\mathcal{I}, w) + 2$ bins. ◁

For every constant $\varepsilon > 0$, it also holds that Algorithm 2 runs in polynomial time. Thus, by Claim 24, it follows that Algorithm 2 is an APTAS for 2VBP. ◀

5 Concluding Remarks

In this paper we present a randomized $\left(1 - \frac{\ln 2}{2} - \varepsilon\right) \approx 0.653$-approximation algorithm for 2VMK, for every fixed $\varepsilon > 0$, thus improving the ratio of $(1 - e^{-1} - \varepsilon) \approx 0.632$, which follows from the results of [8] for the SEPARABLE ASSIGNMENT PROBLEM. To the best of our knowledge, this work is the first direct study of 2VMK in the arena of approximation algorithms.

As an interesting direction for future work, we note that our approach, which combines a technique of [8] with a solution for a residual (1-dimensional) MK instance, does not scale to higher dimensions. Specifically, for dVMK instances where $d \geq 3$, the residual algorithm will obtain marginal profit of $\frac{1}{d \cdot m} \cdot \mathrm{OPT}(\mathcal{I})$ per configuration, which is always lower than the marginal profit obtained by the randomized rounding of [8], due to (4). Hence, for $d \geq 3$, a better approximation ratio is achieved by random sampling of the whole solution as in [8]. We believe that this bottleneck can be resolved by an iterative randomized rounding approach, similar to the approach used in [13] for 2-DIMENSIONAL VECTOR BIN PACKING. This approach can potentially lead also to an improved approximation for 2VMK.

References

1 Max Bannach, Sebastian Berndt, Marten Maack, Matthias Mnich, Alexandra Lassota, Malin Rau, and Malte Skambath. Solving Packing Problems with Few Small Items Using Rainbow Matchings. In *Proc. of MFCS*, pages 11:1–11:14, 2020.

2 Nikhil Bansal, Alberto Caprara, and Maxim. Sviridenko. A new approximation method for set covering problems, with applications to multidimensional bin packing. *SIAM Journal on Computing*, pages 1256–1278, 2010.

3 Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. A sharp concentration inequality with applications. *Random Structures & Algorithms*, 16(3):277–292, 2000.

4 Valentina Cacchiani, Manuel Iori, Alberto Locatelli, and Silvano Martello. Knapsack problems-an overview of recent advances. part ii: Multiple, multidimensional, and quadratic knapsack problems. *Computers & Operations Research*, 2022.

5 Ricardo Stegh Camati, Alcides Calsavara, and Luiz Lima Jr. Solving the virtual machine placement problem as a multiple multidimensional knapsack problem. *ICN 2014*, 264, 2014.

6 Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.

7 Tome Cohen, Ariel Kulik, and Hadas Shachnai. Improved approximation for two-dimensional vector multiple knapsack. *arXiv preprint*, 2023. `arXiv:2307.02137`.

8 Lisa Fleischer, Michel X Goemans, Vahab S Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum separable assignment problems. *Mathematics of Operations Research*, 36(3):416–431, 2011.

9 Alan M Frieze, Michael RB Clarke, et al. Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses. *European Journal of Operational Research*, 15(1):100–109, 1984.

10 Klaus Jansen. Parameterized approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 39(4):1392–1412, 2010.

11 Klaus Jansen. A fast approximation scheme for the multiple knapsack problem. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 313–324, 2012.

12 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

13 Ariel Kulik, Matthias Mnich, and Hadas Shachnai. Improved approximations for vector bin packing via iterative randomized rounding. In *Proc. of FOCS (to appear)*, 2023.

14 Ariel Kulik and Hadas Shachnai. There is no EPTAS for two-dimensional knapsack. *Information Processing Letters*, 110(16):707–710, 2010.

15 Alexandra Lassota, Aleksander Łukasiewicz, and Adam Polak. Tight Vector Bin Packing with Few Small Items via Fast Exact Matching in Multigraphs. In *Proc. of ICALP*, pages 87:1–87:15, 2022.

16 Colin McDiarmid et al. On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188, 1989.

17 Arka Ray. There is no APTAS for 2-dimensional vector bin packing: Revisited. *arXiv preprint*, 2021. `arXiv:2104.13362`.

18 Yang Song, Chi Zhang, and Yuguang Fang. Multiple multidimensional knapsack problem and its applications in cognitive radio networks. In *MILCOM 2008-2008 IEEE Military Communications Conference*, pages 1–7. IEEE, 2008.

19 Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.

20 Jan Vondrák. A note on concentration of submodular functions. *arXiv preprint*, 2010. `arXiv:1005.2791`.

21 Gerhard J Woeginger. There is no asymptotic PTAS for two-dimensional vector packing. *Information Processing Letters*, 64(6):293–297, 1997.

A Compact DAG for Storing and Searching Maximal Common Subsequences

Alessio Conte
Università di Pisa, Italy

Roberto Grossi
Università di Pisa, Italy

Giulia Punzi
National Institute of Informatics, Tokyo, Japan

Takeaki Uno
National Institute of Informatics, Tokyo, Japan

Abstract

Maximal Common Subsequences (MCSs) between two strings X and Y are subsequences of both X and Y that are maximal under inclusion. MCSs relax and generalize the well known and widely used concept of Longest Common Subsequences (LCSs), which can be seen as MCSs of maximum length. While the number both LCSs and MCSs can be exponential in the length of the strings, LCSs have been long exploited for string and text analysis, as simple compact representations of all LCSs between two strings, built via dynamic programming or automata, have been known since the '70s. MCSs appear to have a more challenging structure: even listing them efficiently was an open problem open until recently, thus narrowing the complexity difference between the two problems, but the gap remained significant. In this paper we close the complexity gap: we show how to build DAG of polynomial size – in polynomial time – which allows for efficient operations on the set of all MCSs such as enumeration in Constant Amortized Time per solution (CAT), counting, and random access to the i-th element (i.e., rank and select operations). Other than improving known algorithmic results, this work paves the way for new sequence analysis methods based on MCSs.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Information systems → Structured text search

Keywords and phrases Maximal common subsequence, DAG, Compact data structures, Enumeration, Constant amortized time, Random access

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.21

Related Version *Previous Version*: `https://arxiv.org/abs/2307.13695`

Funding *Alessio Conte*: Partially supported by MUR PRIN Project n. 2022TS4Y3N – EXPAND.
Roberto Grossi: Work partially supported by MUR PRIN Project n. 2022TS4Y3N – EXPAND.
Giulia Punzi: Work partially supported by JSPS KAKENHI Grant Number JP20H05962.
Takeaki Uno: Work partially supported by JSPS KAKENHI Grant Numbers JP20H05962, JP20H00595.

Acknowledgements We thank the anonymous Referees for their comments, leading us to the current version of Theorem 13.

1 Introduction

The *Longest Common Subsequence* (LCS) [4, 12, 16, 25] have thoroughly been studied in a plethora of string comparison application domains, like spelling error correction, molecular biology, and plagiarism detection, to name a few. The LCS is a special case of *Maximal Common Subsequence* (MCS) for any two strings X, Y: it is a string S that is a subsequence of both X and Y, and is inclusion-maximal, namely, no other string S' containing S is

also a common subsequence of X and Y. The set of all MCSs for X and Y is denoted by $MCS(X,Y)$. For example, $MCS(X,Y) = \{\texttt{TACA}, \texttt{G}\}$ for $X = \texttt{TCACAG}$ and $Y = \texttt{GTACTA}$, whereas $\texttt{TCA} \notin MCS(X,Y)$ as it is contained in $\texttt{TACA}$, and the latter is also the only LCS. A crucial observation is the following: if S is a common subsequence, it is always contained in some MCS but this is not necessarily true for LCS (e.g. $S = \texttt{G}$ is not contained in $\texttt{TACA}$).

In general, LCSs only provide us with information about the longest possible alignment between two strings, while MCSs offer a different range of information, possibly revealing slightly smaller but alternative alignments. For very long strings an MCS may be just slightly shorter than an LCS but provide information on parts of the strings not found by LCSs. In principle, MCS could provide helpful information in all the applications where LCS are used.

While there is a quadratic conditional lower bound for the computation of LCS, based on the Strong Exponential Time Hypothesis [1], no such bound exists for MCS: actually, an MCS between two strings of length n can be extracted in $O(n\sqrt{\log n / \log\log n})$ time [22]. Moreover it is NP-hard to compute the LCS of an arbitrary number of strings [15], whereas a recent polynomial-time algorithm for extracting an MCS from multiple strings exists [11]. It is worth noting that even though there are a few more different approaches in the literature to find LCS or common subsequences with some kind of constraints (e.g. common subsequence trees [13], common subsequence automata [7]), the above observations motivate further investigation to directly deal with MCS.

In this paper we proceed along that direction, and consider the problem of storing and searching the set $MCS(X,Y)$. The main hurdle is that $MCS(X,Y)$ could contain an exponential number of distinct strings [6]: consequently, any trie-based or immediate automaton representation would require *exponential* time and space for its construction. We improve significantly over this direction as we list in our contributions, where $n = \max\{|X|, |Y|\}$ and σ is the size of the alphabet Σ for X and Y:

- We introduce a labeled compact direct acyclic graph $\mathsf{MDAG}(X,Y)$ (for *MCS DAG*) that represents the strings in $MCS(X,Y)$ in polynomial space $O(n^3\sigma)$ and can be built in polynomial time $O(n^3\sigma \log n)$. The previously mentioned conditional lower bound for LCS applies, since the longest path here is an LCS of X and Y.
- For any string P, we show how $\mathsf{MDAG}(X,Y)$ can *search* the strings with prefix P from $MCS(X,Y)$ and report them in lexicographic order, in $O(|P| \log \sigma + occ)$ time, where occ is the number of reported strings.
- We can list all the strings from $MCS(X,Y)$ lexicographically in constant amortized time (CAT), namely, $O(|MCS(X,Y)|)$ time.
- By adding $O(n)$-bits per node, we show how $\mathsf{MDAG}(X,Y)$ with input i (where $1 \le i \le |MCS(X,Y)|$) can *select* the ith string S in lexicographic order from $MCS(X,Y)$, in $O(|S| \log \sigma)$ operations on $O(n)$-bit integers; the inverse operation of *rank* for input S is supported in the same complexity, where i is returned.

In Section 3 we implement $\mathsf{MDAG}(X,Y)$ as a direct acyclic graph (DAG) where the only zero indegree node is the source s and the only zero outdegree node is the target t. Each edge is labeled with a character from the alphabet, so that any two outgoing edges from the same node have different characters as labels. Moreover, each st-path corresponds to a string in $MCS(X,Y)$, obtained by concatenating the characters on the edge traversed along the st-path; vice versa, each string in $MCS(X,Y)$ is spelled out by an st-path. In order to define and build $\mathsf{MDAG}(X,Y)$ we use some properties on MCSs previously introduced in [6] and described in Section 2, plus some new ideas to keep the size of MDAG polynomial. As a result, after MDAG has been built and its unary paths have been compacted, the aforementioned operations can be implemented in a simple way, as discussed in Section 4.

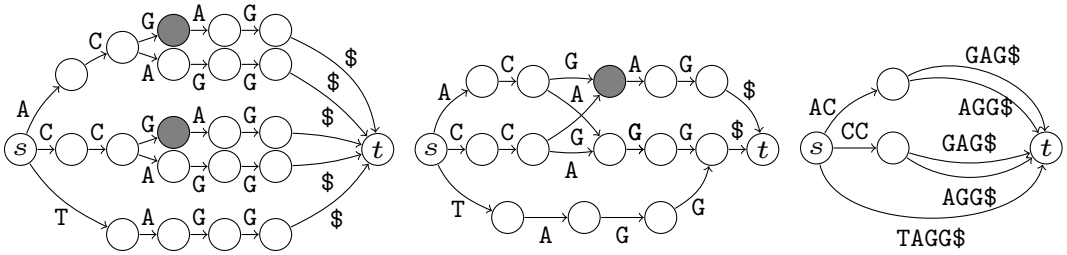


Figure 1 Left: edge-labeled DAG for strings $X = \texttt{TCACAGAGA}$ and $Y = \texttt{ACCCGTAGG}$, where each st-path corresponds to a string in $MCS(X, Y) = \{\texttt{ACAGG}, \texttt{ACGAG}, \texttt{CCAGG}, \texttt{CCGAG}, \texttt{TAGG}\}$. Center: MDAG for the same strings X and Y; for instance, the two shaded nodes in the left graph were compacted into one, shaded in the right. Right: its compact MDAG, where $MCS(X, Y)$ is incrementally output as $push(\texttt{AC})$, $push(\texttt{AGG})$, $pop()$, $push(\texttt{GAG})$, $pop()$, $pop()$, $push(\texttt{CC})$, $push(\texttt{AGG})$, $pop()$, $push(\texttt{GAG})$, $pop()$, $pop()$, $push(\texttt{TAGG})$, $pop()$, and constant-space references to string labels are the arguments of $push()$. Note that before each $pop()$ immediately following a $push()$, we find on the stack the string labels whose concatenation gives a string from $MCS(X, Y)$.

An example of MDAG and compact MDAG is shown in Figure 1, where $MCS(X, Y)$ is enumerated in CAT using constant-space references to the strings that label the edges in $\mathsf{MDAG}(X, Y)$, by simply using a stack [23]. As formalized by Frank Ruskey [20, Section 1.7] in the "Don't Count the Output Principle", we account for the amount of data change that takes place, and the latter takes CAT per solution.

Related work. Maximal common subsequences were first introduced in [9], in the context of LCS approximation. The first algorithm for finding an MCS between two strings was presented in [21], and subsequently refined in [22]. The latter algorithm finds an MCS between two strings of length n in $O(n\sqrt{\log n / \log\log n})$. These algorithms can be also used to extend a given sequence to a maximal one in the same time, and to check whether a given subsequence is maximal in $O(n)$ time. In [11] the authors considered the problem of finding an MCS of $m > 2$ strings of total length n, and were able to solve this in $O(mn \log n)$ time and $O(n)$ space. As for MCS enumeration, [6] showed that this task can be performed in $O(n \log n)$ delay and quadratic space.

The automaton approach has been used in literature to deal with subsequence-related problems. The Directed Acyclic Subsequence Graph (DASG) introduced in [3] is an automaton which accepts all subsequences of a given string S. Given a set of strings, it can also be generalized to accepting subsequences of any string in the set. Later, the common subsequence automata (CSA) was introduced [7, 8, 24], which instead accepts *common* subsequences of a set of strings. Such automaton is similar to the common subsequence tree of [13], and it can also be used to find a longest common subsequence between two strings [17]. Binary decision diagrams such as ZDD [18] and SeqBDD [14] could potentially be employed to compactly store $MCS(X, Y)$ but their worst-case behaviour typically involves exponential construction time and space.

We stress that it is not straightforward to efficiently adapt all of these structures to the MCS problem: indeed, figuring out which of the accepted strings are subsequences of each other is not an immediate task. We therefore opted for a different approach, directly defining and constructing an MCS automaton, based on combinatorial properties specific to MCSs (during the review period of this manuscript, a pre-print on MCS automata appeared [10].)

Preliminaries. [*String notation*] A string S over an alphabet Σ (of size $\sigma = |\Sigma|$) is a concatenation of any number of its characters. The empty string is denoted with ε. For $i \in [0, |S| - 1]$, character $S[i]$ occurs at position i of string S, and position $next_S(c, i)$ denote the next occurrence of character c after position i in string S, if it exists (otherwise, $next_S(c, i) = |S| - 1$). The notation $S_{<i}$ indicates $S[0, i - 1]$, and $S_{\leq i}$ indicates $S[0, i]$. We say that S is a *subsequence* of a string X, denoted $S \subset X$, if there exist indices $0 \leq i_0 < ... < i_{|S|-1} < |X|$ such that $X[i_k] = S[k]$ for all $k \in [0, |S| - 1]$. In this case we also say that X *contains* S. Given two strings X and Y, a string S is a common subsequence if $S \subset X$ and $S \subset Y$. Furthermore, S is a *Maximal Common Subsequence*, denoted $S \in MCS(X, Y)$, if it is a common subsequence which is *inclusion-maximal*: there is no string $T \neq S$ such that $S \subset T \subset X, Y$. [*Graph notation*] Given a directed graph $G = (V, E)$, we define the in-neighbors of a node $v \in V$ as $N^-(v) = \{u \in V \mid (u, v) \in E\}$, and the out-neighbors as $N^+(v) = \{z \in V \mid (v, z) \in E\}$; the in-degree is $d^-(v) = |N^-(v)|$ and the out-degree is $d^+(v) = |N^+(v)|$. A subgraph of graph $G = (V, E)$ is a graph $H = (W, F)$ such that $W \subseteq V$ and $F \subseteq E$. A *path* P in a graph $G = (V, E)$ is a sequence of distinct adjacent nodes: $P = u_1...u_k$ where $u_i \in V$ for all i, $u_i \neq u_j$ for all $i \neq j$, and $(u_i, u_{i+1}) \in E$ for all i. P is called an u_1u_k-path. A path where the first and last nodes coincide is called a *cycle*. A directed graph with no cycles is called a directed acyclic graph, or *DAG*.

2 Structure of MCSs

In this section, we show known properties of MCSs which will be at the base of MDAG. Firstly, we show how to naturally represent all the strings in $MCS(X, Y)$ for any two given strings X and Y, using a finite-state automaton A corresponding to a DAG with a single source s and a single target t, where each edge is labeled with a character of the alphabet Σ; we show that st-paths in A have a one-to-one correspondence with the strings in $MCS(X, Y)$. Secondly, we recall and contextualize some useful properties of MCSs proven in [6].

2.1 Modeling MCS as an Exponentially Large DAG

We here define a deterministic finite-state automaton A that accepts the strings in $MCS(X, Y)$. It is more convenient to define it directly as an equivalent DAG, using the extended alphabet $\Sigma' = \Sigma \cup \$$, where the dollar sign is a new special character to identify the end of an MCS.

▶ **Definition 1.** *The edge-labeled DAG $A = (V \cup \{s, t\}, E, \ell(\cdot))$ is defined for two input strings X and Y to store their set $MCS(X, Y)$, as follows:*

- *Every node in $V \cup \{s\}$ corresponds to a distinct prefix of a string in $MCS(X, Y)$, with s called* source node *corresponding to the empty string prefix.*
- *For any two nodes $u, u' \in V$, let P, P' be their corresponding two prefixes, and let $c \in \Sigma$ be a character. Then, $P' = P\,c$ iff $(u, u') \in E$ with label $\ell(u, u') = c$.*
- *For any node $u \in V$, if its corresponding prefix is a whole string $P \in MCS(X, Y)$, then $(u, t) \in E$ with label $\ell(m, t) = \$$.*

Given DAG A, we have a bijection between the strings in $MCS(X, Y)$ and the labeled st-paths (see the left of Figure 1 for an example). Indeed, it is immediate that any such path corresponds to a string in $MCS(X, Y)$ due to the way edges are placed. Vice versa, by definition, we cannot have two out-going edges from the same node that share the same label. Therefore, each MCS has exactly one corresponding st-path. Note that the property of the outgoing labels also implies that $d^+(u) \leq \sigma$ for all $u \in V$, and therefore $|E| \leq \sigma|V|$. As the number of nodes in A is $\Omega(|MCS(X, Y)|)$, its size is exponential in the worst case.

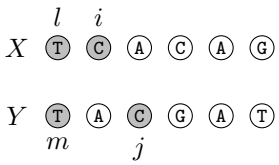
▶ Remark 2. A can be seen as an acyclic deterministic finite-state automaton that accepts the set $MCS(X, Y)$. We have one state for each prefix of a string in $MCS(X, Y)$. Given two states q, q', respectively corresponding to prefixes P, P' of strings in $MCS(X, Y)$, the transition function δ is given by $\delta(q, c) = q'$ if and only if $P' = P\,c$.

2.2 Concepts Borrowed from MCS Enumeration

We now present a summary of some concepts introduced in [6], which will be used for our results. Let string P be called a *valid prefix* if there exists a string Q such that their concatenation is $PQ \in MCS(X, Y)$. Given a valid prefix P, the set of characters c such that $P\,c$ is still a valid prefix are called *valid extensions*.

Finding valid extensions of P is not straightforward. Let P be a valid prefix, and let $X_{\leq l}$ and $Y_{\leq m}$ be respectively the shortest prefixes of X and Y that contain P as a subsequence. Consider a pair of positions $i > l$ and $j > m$, respectively in X and Y, corresponding to the same character, say, $c \in \Sigma$. In order to assess whether c is a valid extension, checking if $P\,c$ is a maximal common subsequence of $X_{\leq i}$ and $Y_{\leq j}$ is necessary, but not sufficient:

▶ **Example 3.** Consider $X = \texttt{TCACAG}$ and $Y = \texttt{TACGAT}$, with $MCS(X, Y) = \{\texttt{TACA}, \texttt{TACG}\}$.



Consider the valid prefix T, for which $l = 0$ and $m = 0$. Let $i = 1$ and $j = 2$: these correspond to character C, and clearly $\texttt{TC} \in MCS(X_{\leq i}, Y_{\leq j}) = MCS(\texttt{TC}, \texttt{TAC})$. Still, C is not a valid extension of valid prefix T, as TC is not a prefix of any MCS.

To circumvent this problem, the set $Ext_{l,m}$ of *candidate extensions* is defined in [6]: this is a set of pairs of positions (i, j), with $i \geq l$ and $j \geq m$, whose definition relies solely on the pair (l, m). The membership of (i, j) to $Ext_{l,m}$ completes the characterization of valid extensions (see Theorem 3 in [6]):

- Let P be a valid prefix, and let $X_{\leq l}$ and $Y_{\leq m}$ be respectively the shortest prefixes of X and Y that contain P as a subsequence. Note that $P \in MCS(X_{\leq l}, Y_{\leq m})$.
- Then $P\,c$ is a valid prefix if and only if the following two conditions hold:
  1. There exists i and j such that $c = X[i] = Y[j]$ and $P \in MCS(X_{<i}, Y_{<j})$;
  2. This pair of positions satisfies $(i, j) \in Ext_{l,m}$.

More details on the construction of set $Ext_{l,m}$ are beyond the scope of this paper, but they can be found in [6, Section 2.3], along with a $O(\sigma \log n)$ time method for its computation. In Example 3, it is $(i, j) \notin Ext_{l,m}$.

The notion of *swings* is a key concept to quickly verify the first condition above. Swings characterize the amount by which we can "move" a given character occurrence while retaining maximality (see Figure 2, right). Let $X_{\leq l}$ and $Y_{\leq m}$ be respectively the shortest prefixes of X and Y that contain P as a subsequence. The *swing* of P, denoted $\ltimes(P)$, is a pair of integers $\ltimes_T(P)$ and $\ltimes_B(P)$, called respectively *top* and *bottom* swings, given by

- $\ltimes_T(P) = \min\{i > l \mid P \notin MCS(X_{\leq i}, Y_{\leq m})\}$
- $\ltimes_B(P) = \min\{j > m \mid P \notin MCS(X_{\leq l}, Y_{\leq j})\}$.

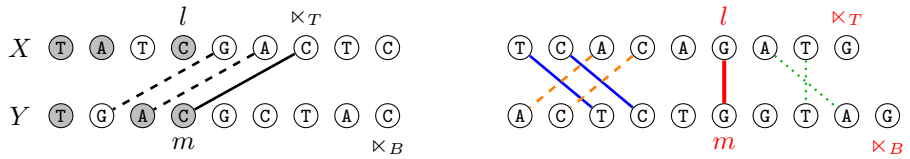


Figure 2 Left: Swings $(\ltimes_T, \ltimes_B)$ for valid prefix `TAC` in strings $X =$ `TATCGACTC` and $Y =$ `TGACGCTAC`. Consider swing $\ltimes_T$: `TAC` $\notin MCS(X_{\leq \ltimes_T}, Y_{\leq m})$ since `TGAC` is a common subsequence (dashed). Note how the bottom swing $\ltimes_B$ is not given by the next occurrence of `C` after m, since `TAC` is still maximal for strings `TATC` and `TGACGC`. Right: Two prefixes, `TCG` (solid blue) and `ACG` (dashed orange), both ending at solid red positions (l, m), and having the same swings $(\ltimes_T, \ltimes_B)$. The valid extensions are the same (dotted green): `A` and `T`.

It follows from the definition that $P \in MCS(X_{<i}, Y_{<j}) \iff i \leq \ltimes_T(P)$ and $j \leq \ltimes_B(P)$. Given the swings, condition 1 can thus be checked in constant time [6]. Note that the definition of top swing necessarily implies $X[i] = X[l]$.[1] The symmetric holds for the bottom swing.

We briefly also recall how to incrementally compute the swings of a prefix, first described in [6], as it will be useful in our proofs. We only describe the procedure for the top swing as the bottom swing is symmetrical:

- If P is composed of a single character c, with first occurrence in X at position l and in Y at position m, then it suffices to compute, for every character d in $Y_{<m}$, the first occurrence of d in $X_{>l}$, and take the minimum of these. The swing then corresponds to the first occurrence of c after such minimum.
- Let $P = p_1 \cdots p_N$ be a valid prefix with $N > 1$, and let $l_1, ..., l_N$ (resp. $m_1, ..., m_N$) be the positions of X (resp. Y) such that $X_{\leq l_i}$ (resp. $Y_{\leq m_i}$) is the shortest prefix containing $p_1 \cdots p_i$. The *personal top swing* $\ltimes_T(l_N, m_N)$ of the last position is the top swing of character p_N when seen as a prefix over the strings $X_{>l_{N-1}}, Y_{>m_{N-1}}$, instead of over the whole strings (and thus computed as above). The personal bottom swing is defined analogously. In other words, the personal swing of a character expresses the change necessary to have an insertion between itself and the previous character of the prefix (see Figure 3 for an example). The top swing of P is the minimum between the personal swing of (l_N, m_N), and the first occurrence of $Y[m_N] = p_N$ after the top swing of prefix $p_1 \cdots p_{N-1}$. This second swing indicates the change required for an insertion in the previous part of the prefix.

Updating the swings when a character is added can be done in $O(\sigma)$ time, provided that we can find in constant time the next occurrence of a character c after a given position in the strings; more importantly, this update does *not* need knowledge of the whole prefix, but just the positions its final character (l_N, m_N) and their current swings.

[1] Consider any $h > l$ such that $P \notin MCS(X_{\leq h}, Y_{\leq m})$. Since $P = p_1 \cdots p_s \in MCS(X_{\leq l}, Y_{\leq m})$ and we are only extending string X, P can only become non-maximal if an insertion occurs between p_i and p_{i+1}, for $i < s$. If h' is the last occurrence of $p_s = X[l]$ before h, it is $P \notin MCS(X_{\leq h'}, Y_{\leq m})$, as the substring of X between h' and h does not contain a suffix of P as subsequence. The minimum index $i > l$ such that $P \notin MCS(X_{\leq i}, Y_{\leq m})$ must satisfy $X[i] = X[l]$.

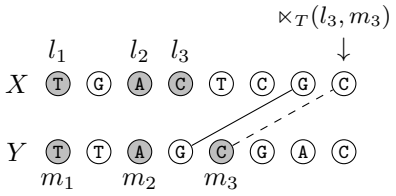


Figure 3 Personal top swing for edge (l_3, m_3), corresponding to the last character of the leftmost mapping of prefix `TAC`. Indeed, such swing allows for insertion of character `G` after (l_2, m_2).

3 Polynomial-Size MDAG

The construction of DAG A satisfying the conditions of Definition 1 would require exponential time and space: the number of nodes of A is between $\Omega(|MCS(X,Y)|)$ and $O(n|MCS(X,Y)|)$, so it can be exponential in n. In this section, we show how to obtain MDAG, where we still have a bijection between st-paths and MCS, but which can instead always be constructed in $O(n^3\sigma \log n)$ time and $O(n^3\sigma)$ space, as per Theorem 13. Intuitively, the relevant information discussed for A are the quadruples (l, m, t, b), where $X_{\leq l}$ and $Y_{\leq m}$ are some prefixes and pair t, b is some swing: these quadruples are the candidates for being nodes in MDAG.

The formal definition of MDAG is based on an equivalence relation over the nodes of A, given in Section 3.1. Afterwards, in Section 3.2, we describe an algorithm for *directly* constructing MDAG. We present the complexity bounds for the construction in Section 3.3.

3.1 Equivalence Relation for Defining MDAG

Let A be a DAG as defined in Definition 1. Our construction algorithm for MDAG is based on the concepts from Section 2.2. The idea is to use the characterization of valid extensions to identify the out-neighbors of a given node of DAG A. We identify an equivalence relation over the prefixes of $MCS(X,Y)$, and thus on the nodes of A, that allow us to always build MDAG in polynomial time and space. We begin with the following lemma:

▶ **Lemma 4.** *Given any valid prefix P, let $X_{\leq l}$ and $Y_{\leq m}$ be the shortest prefixes containing P, and $\ltimes(P) = \langle t, b\rangle$ be its swing. Consider another valid prefix $P' \neq P$ with the* same *shortest prefixes $X_{\leq l}, Y_{\leq m}$ and swing $\ltimes(P') = \ltimes(P)$ as P. Then, the set of valid extensions is the same for both P and P', and for each valid extension $c \in \Sigma$, the swings of Pc are the same as the ones of $P'c$.*

Proof. The definition of $Ext_{l,m}$ only depends on the value of l and m, therefore such set is the same for both P and P'. Since the swings for P and P' are equal, the set of valid extensions is necessarily the same. Let now $c \in \Sigma$ be a valid extension for P and P'. Let $X_{\leq l_c}$ and $Y_{\leq m_c}$ respectively be the shortest prefixes of X and Y containing Pc. These are also the shortest prefixes containing $P'c$: the shortest prefixes containing P and P' were the same, and l_c is simply the first occurrence of c after l, analogously for m_c and m. The swings of Pc are given by the minimum of the swings of P, and the personal swing $\ltimes(l_N, m_N)$ obtained by adding the new character c. The latter personal swing is the same for both Pc and $P'c$, since we are considering the same positions l_c, m_c. Since the previous swings where also equal, this means that the swings of Pc and $P'c$ are indeed the same. ◀

Lemma 4 has an implication for prefixes P and P' that share the *same* swing: if $M_1, ..., M_N$ are strings extending as $PM_i \in MCS(X,Y)$, and $M'_1, ..., M'_M$ extending as $P'M'_i \in MCS(X,Y)$, then they are equal: $\{M_i \mid i = 1, ..., N\} = \{M'_i \mid i = 1, ..., M\}$.

Given a node u of A, let P be the corresponding prefix. We assign u the quadruple of parameters $ID(u) = \langle l, m, \ltimes_T(P), \ltimes_B(P)\rangle$, where l and m are such that $X_{\leq l}$ and $Y_{\leq m}$ are the shortest prefixes containing P. By Lemma 4, this tuple completely identifies the valid extensions of P, which means that it completely identifies the neighbors of node u.

▶ **Corollary 5.** *Let $u \neq u'$ with $ID(u) = ID(u')$. Then, for each $v \in N^+(u)$ there exists exactly one $v' \in N^+(u')$ such that $ID(v) = ID(v')$ and $\ell(u, v) = \ell(u', v')$.*

Therefore, we can define the following *equivalence relation* on the nodes of A: $u \sim u'$ if and only if $ID(u) = ID(u')$. We can then identify a class of equivalent nodes in the DAG, choosing one representative for it. Because of Corollary 5, this does not change the set of labeled *st*-paths of the DAG: the nodes that are identified as one have the same labelled out-edges, leading to the same out-neighbors. Our data structure MDAG is then defined as the DAG resulting from this identification:

▶ **Definition 6.** *Data structure MDAG is a node- and edge-labelled DAG built as follows:*

1. *Start from DAG A (Definition 1). For each node u, consider its (unique) corresponding prefix P, and let $X_{\leq l}$ and $Y_{\leq m}$ be the shortest prefixes of X and Y containing P, and $\ltimes(P) = (t, b)$. Assign to node u the node-label $ID(u) = \langle l, m, t, b\rangle$.*
2. *Merge every pair of nodes $u \neq u'$ with the same label $ID(u) = ID(u')$ into one node.*

An example of such DAG is shown in the right of Figure 1.

It is possible to further compress the MDAG, as detailed in the next section, by compacting nodes of out-degree 1. This will not change its worst-case size, but will impact the efficiency of our method.

3.2 Direct and Incremental Construction of MDAG

We build MDAG directly, without the intermediate DAG A. We apply the incremental procedure below, in a DFS fashion, using the node IDs to avoid repeated computation. At any moment we have built a node- and edge-labeled DAG $H = (W, F)$. A node $u \in W$ corresponds to a set of prefixes $P_1, ..., P_k$, given by the concatenation of the edge-labels of all *su*-paths using edges of F. All prefixes P_i share the same ending positions of the shortest prefixes of X and Y that contain them, and the corresponding swings; these four values form the label $ID(u)$ assigned to u.

Every recursive call BUILDDAG(u) takes as input a node u which belongs to the current DAG H, and expands DAG H accordingly as follows:

1. Let $ID(u) = \langle l, m, t, b\rangle$. First, compute set $Ext_{l,m}$, and use it to compute the valid extensions: select characters c that have a corresponding pair $(i, j) \in Ext_{l,m}$, with $i \leq t$ and $j \leq b$.
2. For such character c, compute the positions (l_c, m_c) such that $X_{\leq l_c}$ and $Y_{\leq m_c}$ are the shortest prefixes containing Pc, and update the swings t_c, b_c.
3. Now, check if the DAG H generated so far already has a node with label $\langle l_c, m_c, t_c, b_c\rangle$:
   a. If such a node $v \in W$ exists, then simply add edge (u, v) with label c to the edges F of H, without recursing. Indeed, a recursive call for v has been previously performed.
   b. Otherwise, add node v to W, with $ID(v) = \langle l_c, m_c, t_c, b_c\rangle$, and add edge (u, v) to F, with label c. Then, perform the recursive call BUILDDAG(v).

▶ **Corollary 7** (Correctness). *BUILDDAG(s) correctly builds MDAG(X, Y) starting from $H = (\{s\}, \emptyset)$.*

Proof. We show that, at every step, H is a subgraph of DAG A where nodes with equal $ID()$ values have been identified. This is true at the beginning, when $H = \{s\}$. If this holds at the beginning of a recursive call for a node u, then it must hold at the end. Indeed, we use valid extensions to identify neighbors of a given node, which is also the definition of neighbors for a node of A. For each valid extension c, we compute the label $\langle l_c, m_c, t_c, b_c\rangle$ of the corresponding node. Now, if there already exists a node v with such label, we add an edge between u and v, with label c. This correctly identifies the two nodes with equal labels as being the same node, as per operation 2. Otherwise, the new node must be added, with the correct label $\langle l_c, m_c, t_c, b_c\rangle$ as per operation 1, since it represents a new prefix. ◀

Once we have built MDAG, we can easily compute compact MDAG in linear time in the size of MDAG, in the following way. Let us proceed in topological order of the nodes; when a node v with $N^+(v) = \{w\}$ (i.e. out-degree 1) is encountered, we perform the following:

1. For each $u \in N^-(v)$, remove edge (u, v) and add edge (u, w) with label $\ell(u, v)\ell(v, w)$.
2. After all in-neighbors have been processed, remove node v and edge (v, w) from MDAG.

Clearly, the minimum out-degree of is 2, and nodes s and t are never removed.

Complexity. Let us now study the time and space complexity of the procedure. As mentioned before, finding $Ext_{l,m}$ requires $O(\sigma \log n)$ time (see [6] for details). Checking whether each c corresponding to an element of $Ext_{l,m}$ satisfies the swings' condition requires constant time per character. For each such c, computing positions (l_c, m_c) can be done in constant time, using appropriate data structures as outlined next. In order to attain our goal, we only need to ensure constant-time queries for the next occurrence of character c after a given position i. To this end, let us keep two bit-vectors for each $c \in \Sigma$, one for X and one for Y, indicating the positions in which c occurs in the strings. By equipping these vectors with rank and select data structures, which employ $O(n\sigma)$ space, we can find in constant time the next occurrence of any character after a given position [19]. Performing this operation in constant time also allows us to update the swings in $O(\sigma)$ time, as we have explained in Section 2.2. All operations described so far are performed exactly once per node. Therefore, the total time required for Steps 1 and 2 is $O(|V|\sigma \log n)$.

Let us now consider Step 3. We need to check if a node v belongs to the current DAG, and add it if it does not. To be able to efficiently perform these operations, let us keep and dynamically update a bit matrix for pairs l, m, where 1 occurs if that pair currently corresponds to at least one node. Then, each cell filled with a one has an associated balanced binary search tree, which indexes the pairs of swings (t, b) such that there currently is a node $u \in W$ with $ID(u) = \langle l, m, t, b\rangle$. These pairs are ordered according to the total order: $(t, b) < (t', b')$ if and only if $t < t'$ or $t = t'$ and $b < b'$. Lookup and insertions in such data structure require $O(\log n)$ each, and the total space employed is $O(|V|)$. Now, note that we perform a membership check, with subsequent possible insertion, exactly once per edge of the DAG. Recalling that $|E| \leq \sigma|V|$, the total time required for Step 3 is again $O(|E| \log n) = O(|V|\sigma \log n)$. Thus, summing the time required for Steps 1, 2 and 3 yields $O(|V|\sigma \log n)$ total time for the algorithm.

To be able to give final complexity bounds, we therefore need bounds on the size of the DAG we constructed. Trivially, we can bound $|V| = O(n^4)$, since no two nodes share the same ID, and the number of different IDs is bounded by n^4. Therefore, we surely have a polynomial-time and space algorithm for building a DAG. We can actually do better than this: thanks to some properties of the swings, in the next section we show that $|V| = O(n^3)$, leading to the complexity bounds given in Theorem 13.

3.3 Cubic Size of the MCS DAG

To conclude the proof of Theorem 13, we need to study the size of $\mathsf{MDAG}(X, Y)$ as constructed in Section 3.2. We prove a monotonicity property of the swing values, which will allow us to show that the number of nodes of $\mathsf{MDAG}(X, Y)$ is bounded by $O(n^3)$.

In Section 2.2, we saw that the top swing of a valid prefix P is defined as $\ltimes_T(P) = \min\{i > l \mid P \notin MCS(X_{\leq i}, Y_{\leq m})\}$, where $X_{\leq l}$ and $Y_{\leq m}$ are the shortest prefixes of respectively X and Y containing P. In other words, if we start from strings $X_{\leq l}, Y_{\leq m}$ (where P is obviously maximal), it is the minimum extension of string X that ensures at least one insertion in P. Symmetrical definition holds for bottom swings, by switching the two strings.

Recall that, as seen in Section 2.2, if $\ltimes_T(P) = t$ then $X[t] = Y[m] = X[l]$: the swings' positions are occurrences of the last character of the prefix. We also note the following, which follows from the definition of swings:

▶ Remark 8. Let $P = p_1 \cdots p_N$ a valid prefix with swings $\langle t, b\rangle$. Let $X_{\leq l_i}$ and $Y_{\leq m_i}$ be the shortest prefixes respectively of X and Y that contain $p_1 \cdots p_i$. Then, there is at least one match between $Y[m_{N-1}, m_N)$ and $X(l_N, t)$. More specifically, there can either be a match between $Y(m_{N-1}, m_N)$ and $X(l_N, t)$, or between $Y[m_{N-1}]$ and $X(l, t)$ which will lead to an insertion in a previous part of the prefix.

We now present some new swing properties. This lemma proves that, when two prefixes are extended with a valid extension that occurs at the same pair of positions, then the relative order of the swings remains unchanged during the extension:

▶ **Lemma 9.** *Let u and u' be two nodes of MDAG, with $ID(u) = \langle x, y, \ltimes_T(P), \ltimes_B(P)\rangle$ and $ID(u') = \langle x', y', \ltimes_T(P'), \ltimes_B(P')\rangle$ such that $x \neq x'$ or $y \neq y'$, where P (resp. P') is any prefix associated to u (resp. u'). Assume that we have $v \in N^+(u)$ with $ID(v) = \langle l, m, \ltimes_T(P\,c), \ltimes_B(P\,c)\rangle$, and $v' \in N^+(u')$ with $ID(v') = \langle l, m, \ltimes_T(P'\,c), \ltimes_B(P'\,c)\rangle$ (i.e. same positions l, m corresponding to character c). Then, the swings change monotonically:*

- $\ltimes_T(P) < \ltimes_T(P') \Rightarrow \ltimes_T(P\,c) \leq \ltimes_T(P'\,c)$
- $\ltimes_B(P) < \ltimes_B(P') \Rightarrow \ltimes_B(P\,c) \leq \ltimes_B(P'\,c)$

Proof. We prove the result for top swings. Let $\ltimes_T(P) = t$ and $\ltimes_T(P') = t'$, with $t < t'$. We note that for (l, m) to be a valid extension for both prefixes, we must have $l \leq t < t'$ (Swing condition 1 for valid extensions' characterization in Section 2.2). By the incremental computation of swings, the swings of $P\,c$ and $P'\,c$ are computed by taking the minimum between the personal swing $\ltimes_T^{(l,m)}$ of the new positions, and the next occurrence of the corresponding character after the top swings t, t'. More specifically, $\ltimes_T(P\,c) = \min\{\ltimes_T^{(l,m)}, next_X(c, t)\}$ and $\ltimes_T(P'\,c) = \min\{\ltimes_T^{(l,m)}, next_X(c, t')\}$. Since the first component of the minimum is the same, it suffices to prove that $next_X(c, t) \leq next_X(c, t')$ to conclude $\ltimes_T(P\,c) \leq \ltimes_T(P'\,c)$. Indeed, since $t < t'$ are positions in the same string, the next occurrence of a given character after t cannot be strictly bigger than the next occurrence of the same character after t'. Thus, we have proved the claim for top swings; bottom swings are symmetrical. ◀

The next corollary shows that the opposite implication holds for strict inequalities:

▶ **Corollary 10.** *Under the same hypotheses of Lemma 9, we have*

- $\ltimes_T(P\,c) < \ltimes_T(P'\,c) \Rightarrow \ltimes_T(P) < \ltimes_T(P')$
- $\ltimes_B(P\,c) < \ltimes_B(P'\,c)) \Rightarrow \ltimes_B(P) < \ltimes_B(P')$

Proof. We reverse the proof of Lemma 9. Consider top swings, and recall $\ltimes_T(P\,c) = \min\{\ltimes_T^{(l,m)}, next_X(c,t)\}$ and $\ltimes_T(P'\,c) = \min\{\ltimes_T^{(l,m)}, next_X(c,t')\}$, where $t = \ltimes_T(P)$ and $t' = \ltimes_T(P')$. Since the first part of the minimum is the same, and $\ltimes_T(P\,c) < \ltimes_T(P'\,c)$, we have two options

1. $next_X(c,t) \leq \ltimes_T^{(l,m)} \leq next_X(c,t')$, where at most one inequality can be an equality; or
2. $next_X(c,t) < next_X(c,t') \leq \ltimes_T^{(l,m)}$.

In any case, we have $next_X(c,t) < next_X(c,t')$. Since t and t' are positions in the same string, this relationship between the next occurrence of the same character immediately also implies $t < t'$, which concludes the proof. ◀

We are now ready to prove our main result:

▶ **Theorem 11.** *For any two nodes $u \neq u'$ of MDAG, let $ID(u) = \langle l, m, t, b\rangle$ and $ID(u') = \langle l, m, t', b'\rangle$. Then swing pairs for the same l, m do not dominate each other; namely, if $t > t'$, then $b \leq b'$.*

Proof. Consider any $v \in N^-(u)$, and $v' \in N^-(u')$. Let $ID(v) = \langle x, y, t_v, b_v\rangle$ and $ID(v') = \langle x', y', t_{v'}, b_{v'}\rangle$. Let us first assume that $x \neq x'$ or $y \neq y'$, i.e. they are not the same pair of positions. If we look at positions (x, y) and (x', y') we must have either $x \leq x'$ and $y > y'$, or $x > x'$ and $y \leq y'$. Indeed, assume by contradiction that $x \leq x'$ and $y \leq y'$. Since both of these nodes have a valid extension corresponding to positions (l, m), we must also have $x \leq x' < l < t_v, t_{v'}$ and $y \leq y' < m < b_v, b_{v'}$. Then, (l, m) would not be a valid extension for v: the corresponding prefix is not maximal until the positions given by the swings, since we have an insertion corresponding to the character occurring at positions (x', y').

We now show that $t > t'$ implies $y > y'$. By Remark 8, t is the smallest value such that a match occurs between $X[l+1, t)$ and $Y[y, m-1]$. That is, there is no $\tau < t$ such that $X[l+1, \tau)$ and $Y[y, m-1]$ have a match. Let $t > t'$, and assume by contradiction that $y \leq y'$. Then, $Y[y', m-1] \subseteq Y[y, m-1]$. By definition of t', there is a match between $X[l+1, t')$ and $Y[y', m-1] \subseteq Y[y, m-1]$. This is a contradiction on the minimality of t: there is a smaller $\tau = t' < t$ which yields a match. Now, since we cannot have both $y' \leq y$ and $x' \leq x$, we must have $x \leq x'$. By a symmetrical reasoning, we show that the bottom swings must satisfy $b' \leq b$. Indeed, recall that b is the minimum value for which a match occurs between $X[x, l-1]$ and $Y[m+1, b]$, and assume by contradiction that $b > b'$. Since we have $X[x', l-1] \subseteq X[x, l-1]$, we have a match between $X[x, l-1]$ and $Y[m+1, b']$ for a smaller value $b' < b$: contradiction.

Let us now consider the case where the in-neighbors v and v' have the same pair of positions in their IDs: $x = x'$ and $y = y'$. Let us inductively consider $w_{i+1} \in N^-(w_i)$ and $w'_{i+1} \in N^-(w'_i)$, where $w_0 = v$ and $w'_0 = v'$. We stop at the first j such that $ID(w_j) = \langle x_j, y_j, t_j, b_j\rangle$ and $ID(w') = \langle x'_j, y'_j, t'_j, b'_j\rangle$ with $x_j \neq x'_j$ or $y_j \neq y'_j$. Such pair satisfies the conditions of the first part of the proof, since $x_{j+1} = x'_{j+1}$ and $y_{j+1} = y'_{j+1}$ by hypothesis. Nodes w_j and w'_j are obtained by going backwards in the DAG for j steps, starting respectively from nodes u and u'. Since j is the first index such that the corresponding positions for extensions differ, we have $\ell(w_{k+1}, w_k) = \ell(w'_{k+1}, w'_k)$ for all $k = 1, ..., j-1$. By iterating Corollary 10, we thus have that $t > t'$ implies $t_j > t'_j$. By the first part of the proof, we therefore have $b_j \leq b'_j$. By Lemma 9, this propagates to the end of the path in the MCS DAG, to also yield $b \leq b'$. ◀

From Theorem 11, we can derive the following result, which proves that the number of swings for a fixed pair of positions (l, m) is linear:

▶ **Corollary 12.** *For any given choice of l, m, there are just $O(n)$ nodes of MDAG(X, Y) having the form $ID() = \langle l, m, \cdot, \cdot \rangle$.*

Proof. Let us fix l, m, and consider the swings' set $S_{l,m} \subseteq \{0, ..., n-1\} \times \{0, ..., n-1\}$, where $(a, b) \in S_{l,m}$ if and only there exists u such that $ID(u) = \langle l, m, a, b \rangle$. By Theorem 11, if two pairs (a, b) and (c, d) belong to $S_{l,m}$, then it cannot be $a \leq c$ and $c \leq d$ (or vice versa). So one pair cannot dominate the other.

We observe that the size of $|S_{l,m}|$ is the size of the classical Pareto frontier: for an arbitrary set of points in the $\{0, ..., n-1\} \times \{0, ..., n-1\}$ grid, the number of points in a Pareto frontier is less than $2n$. The observation is folklore: each point in the frontier either increases the x-coordinate or decreases the y-coordinate (possibly both). Hence, there cannot be more points on the frontiers as the sum of the n possible x-coordinates plus the n possible y-coordinates. Hence, $|S_{l,m}| = O(n)$. ◀

Therefore, the number of nodes of the MDAG is cubic, as we have $O(n^2)$ choices for l, m, and every such choice gives at most a linear amount of swings (t, b). Furthermore, it is immediate by construction of MDAG that the out-degree of every node is at most σ (the characters that are valid extensions the given prefix). This gives $O(n^3\sigma)$ nodes and edges in MDAG. When obtaining the compact MDAG, the number of nodes and edges do not increase, and a suitable representation of the string labels gives the space occupancy of $O(n^3\sigma)$ memory words stated in Theorem 13.

4 Efficient Operations on MDAG

We describe here how to support some operations on compact MDAG(X, Y), which has source s, target t, and no unary nodes. We assume that each node u stores the number $p(u)$ of ut-paths. As compact MDAG(X, Y) is a DAG of $O(n^3\sigma)$ edges, we can computed $p(u)$ for each node u in total $O(n^3\sigma)$ time by running a DFS, as $p(u)$ is the sum of the $p(v)$'s for the out-neighbors v's of u.

4.1 CAT Enumeration of MCSs

The strings in $MCS(X, Y)$ can be listed in lexicographic order by enumerating the (labeled) st-paths in compact MDAG, which can be done in Constant Amortized Time with a simple DFS algorithm where the out-neighbors of each node are visited in increasing order of the labels of their outgoing edges. Although folklore, for completeness we sketch the DFS algorithm here.

Consider a node u, where initially $u = s$, and denote the set of all ut-paths in $G =$ compact MDAG(X, Y) by $\mathcal{P}_{u,t}(G)$. The central idea is that any ut-path starts with u, followed by an element of $\mathcal{P}_{v,t}(G \setminus u)$, i.e., a path in $G \setminus u$ (i.e. u and its incident edges removed) from an out-neighbor v of u to t. Since G is a DAG, we can go a step further: a path from u cannot reach u again, therefore it is *not* even necessary to remove u (and its incident edges) from the DAG. We can thus represent $\mathcal{P}_{u,t}(G)$ as the following disjoint union: $\mathcal{P}_{u,t}(G) = \bigcup_{v \in N(u)} \{\ell(u, v)P \mid P \in \mathcal{P}_{v,t}(G)\}$, where $\ell(u, v)$ denotes the character(s) labeling edge u, v. From this, it is immediate that enumeration can be performed by keeping a current path prefix which we expand by traversing the DAG, backtracking every time computation terminates for all children of a node.

This recursive procedure runs in constant amortized time. Consider its calls and observe that they form a recursion tree with the following properties:

- Each leaf in the recursion tree is a distinct st-path (and all st-paths are in these leaves without any duplication).
- Each internal node in the recursion tree always generates at least two children, due to the path compression in G: hence the number of internal nodes is not greater than the number of leaves.
- In each node we spend just constant time per recursive call. In each leaf we spend constant time, not accounting for explicitly printing its whole st-path.

Letting N be the number of st-paths, there are N leaves and at most N internal nodes in the recursion tree. Each call takes constant time to generate its node in the recursion tree, which means $O(1)$ time per node/leaf. This gives a total of $O(N)$ time for listing all the st-paths when starting from G, recalling that G has no unary node (except possibly t). This provides a CAT enumeration as we can charge $O(1)$ time per solution to cover this cost.

4.2 Searching, Selecting, and Ranking

We observe that each node u has at most σ out-neighbors and the edges towards them are distinct (each must start with a different character of Σ): this constitutes a "lexicographical partition" of the paths from u to t since, after a common prefix, a path starting with a larger character will lead to a lexicographically larger string.

Searching a string P as a prefix of strings from $MCS(X,Y)$ traverses compact MDAG starting from s and matching the characters in P along the labels for the edges in the path. Either the search fails before reaching the end of P, or it succeeds and leads to a node u (or to an arc with endpoint u). At this point, we can run the CAT enumeration (Section 4.1) starting from u to list all the ut-paths and so all the extensions of P to strings in $MCS(X,Y)$.

Selecting the ith string in lexicographic order from $MCS(X,Y)$ is similar but uses the $O(n)$-bit information $p(v)$ in each traversed node v, that is, $p(v)$ is the number of vt-paths and requires $O(n)$ bits as the number of MCS can be $O(2^n)$. The select operation scans the outgoing edges in order of their labels, and checks the corresponding $p(v)$: whenever the edges scanned have the largest partial sum i, it follows the current edge e and subtracts from i the partial sum of the edges examined before e. It stops at node t. Let S be the string thus found by concatenating the labels on the traced path from s to t. The number of traversed nodes is at most $|S|+1$, and in each node we will consider up to σ edges, for a total cost of $O(|S|\log\sigma)$, as this can be further optimized by storing, for each edge, the sum of the annotations of the previous edges, and using binary search on these $O(\sigma)$ sums. Ranking S is like searching S and using the partial sums as mentioned above to get a total sum of i when t is reached.

▶ **Theorem 13.** *Given two strings X and Y of length n on an alphabet of size σ, compact MDAG(X,Y) stores $MCS(X,Y)$ in space $O(n^3\sigma)$ and can be built in $O(n^3\sigma\log n)$ time. It supports the following operations:*

- *List all strings in $MCS(X,Y)$ in $O(|MCS(X,Y)|)$ time, i.e., Constant Amortized Time.*
- *For a given string P, report just the strings from $MCS(X,Y)$ that have prefix P, in $O(|P|\log\sigma + occ)$ time, where occ is the number of reported strings.*

The $MCS(X,Y)$ can be augmented with $O(n)$ bits per node, so that two further operations are supported, where each operation handles $O(n)$-bit integers:

- *For any integer $1 \le i \le |MCS(X,Y)|$, select the ith string S in lexicographic order from $MCS(X,Y)$, in $O(|S|\log\sigma)$ operations.*
- *For any string $S \in MCS(X,Y)$, return its rank i among the strings in lexicographic order from $MCS(X,Y)$, in $O(|S|\log\sigma)$ operations.*

It is worth noting that (compact) MDAG can be stored in a succinct way, for example, using some recent methods introduced for automata [5].

5 Conclusions

In this paper we considered the problem of storing and searching the Maximal Common Subsequences of two input strings, as they may reveal further common structures in sequence analysis with respect to the LCSs. Our main contribution is that of reducing time and space from exponential bounds, using the current state of the art, to polynomial bounds, using our new compact DAG, which efficiently supports CAT (constant amortized time) enumeration, counting, and random access to the i-th element (i.e., rank and select operations). As future work, we plan to improve the time and space bounds for rank and select in Theorem 13 by storing the number of paths in each node in a more refined way, so as the space does not increase asymptotically, and the time bounds reduce from $O(|S| \log \sigma)$ operations on $O(n)$-bit integers to $O(|S| \log \sigma + n)$ time, borrowing ideas from [2].

References

1 A. Abboud, A. Backurs, and V. V. Williams. Tight hardness results for LCS and other sequence similarity measures. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 59–78, October 2015. doi:10.1109/FOCS.2015.14.

2 Amihood Amir, Gianni Franceschini, Roberto Grossi, Tsvi Kopelowitz, Moshe Lewenstein, and Noa Lewenstein. Managing unbounded-length keys in comparison-driven data structures with applications to online indexing. *SIAM J. Comput.*, 43(4):1396–1416, 2014. doi:10.1137/110836377.

3 Ricardo A Baeza-Yates. Searching subsequences. *Theoretical Computer Science*, 78(2):363–376, 1991.

4 L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000*, pages 39–48, September 2000. doi:10.1109/SPIRE.2000.878178.

5 Sankardeep Chakraborty, Roberto Grossi, Kunihiko Sadakane, and Srinivasa Rao Satti. Succinct representation for (non)deterministic finite automata. *J. Comput. Syst. Sci.*, 131:1–12, 2023. doi:10.1016/j.jcss.2022.07.002.

6 Alessio Conte, Roberto Grossi, Giulia Punzi, and Takeaki Uno. Enumeration of maximal common subsequences between two strings. *Algorithmica*, pages 1–27, 2022.

7 Maxime Crochemore, Bořivoj Melichar, and Zdeněk Troníček. Directed acyclic subsequence graph – Overview. *Journal of Discrete Algorithms*, 1(3-4):255–280, 2003.

8 Maxime Crochemore and Zdeněk Troníček. Directed acyclic subsequence graph for multiple texts. *Rapport IGM*, pages 99–13, 1999.

9 C. B. Fraser, R. W. Irving, and M. Middendorf. Maximal common subsequences and minimal common supersequences. *Information and Computation*, 124(2):145–153, 1996. doi:10.1006/inco.1996.0011.

10 Miyuji Hirota and Yoshifumi Sakai. Efficient algorithms for enumerating maximal common subsequences of two strings. *CoRR*, abs/2307.10552, 2023. doi:10.48550/arXiv.2307.10552.

11 Miyuji Hirota and Yoshifumi Sakai. A fast algorithm for finding a maximal common subsequence of multiple strings. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, page 2022DML0002, 2023.

12 D. S. Hirschberg. Algorithms for the longest common subsequence problem. *J. ACM*, 24(4):664–675, October 1977. doi:10.1145/322033.322044.

13 W. J. Hsu and M. W. Du. Computing a longest common subsequence for a set of strings. *BIT Numerical Mathematics*, 24(1):45–59, 1984.

14 Elsa Loekito, James Bailey, and Jian Pei. A binary decision diagram based approach for mining frequent subsequences. *Knowl. Inf. Syst.*, 24(2):235–268, 2010. doi:10.1007/s10115-009-0252-9.

15 David Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM (JACM)*, 25(2):322–336, 1978.

16 W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1.

17 Bořivoj Melichar and Tomáš Polcar. The longest common subsequence problem a finite automata approach. In *Implementation and Application of Automata: 8th International Conference, CIAA 2003 Santa Barbara, CA, USA, July 16–18, 2003 Proceedings*, pages 294–296. Springer, 2003.

18 Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proceedings of the 30th International Design Automation Conference*, DAC '93, pages 272–277, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/157485.164890.

19 R. Raman, V. Raman, and S. R. 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.

20 Frank Ruskey. Combinatorial generation. *Preliminary working draft. University of Victoria, Victoria, BC, Canada*, 11:20, 2003.

21 Yoshifumi Sakai. Maximal common subsequence algorithms. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*, volume 105 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:10, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CPM.2018.1.

22 Yoshifumi Sakai. Maximal common subsequence algorithms. *Theoretical Computer Science*, 793:132–139, 2019. doi:10.1016/j.tcs.2019.06.020.

23 Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006. Computing and Combinatorics.

24 Zdeněk Troníček. Common subsequence automaton. In *International Conference on Implementation and Application of Automata*, pages 270–275. Springer, 2002.

25 R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974. doi:10.1145/321796.321811.

Prefix Sorting DFAs: A Recursive Algorithm

Nicola Cotumaccio
Gran Sasso Science Institute, L'Aquila, Italy
Dalhousie University, Halifax, Canada

Abstract

In the past thirty years, numerous algorithms for building the suffix array of a string have been proposed. In 2021, the notion of suffix array was extended from strings to DFAs, and it was shown that the resulting data structure can be built in $O(m^2 + n^{5/2})$ time, where n is the number of states and m is the number of edges [SODA 2021]. Recently, algorithms running in $O(mn)$ and $O(n^2 \log n)$ time have been described [CPM 2023].

In this paper, we improve the previous bounds by proposing an $O(n^2)$ recursive algorithm inspired by Farach's algorithm for building a suffix tree [FOCS 1997]. To this end, we provide insight into the rich lexicographic and combinatorial structure of a graph, so contributing to the fascinating journey which might lead to solve the long-standing open problem of building the suffix tree of a graph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Pattern matching

Keywords and phrases Suffix Array, Burrows-Wheeler Transform, FM-index, Recursive Algorithms, Graph Theory, Pattern Matching

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.22

Related Version *Full Version*: `https://arxiv.org/abs/2305.02526`

Funding This work was partially funded by Dante Labs.

Acknowledgements I thank Nicola Prezza for pointing out the paper [22].

1 Introduction

The *suffix tree* [31] of a string is a versatile data structure introduced by Weiner in 1973 which allows solving a myriad of combinatorial problems, such as determining whether a pattern occurs in the string, computing matching statistics, searching for regular expressions, computing the Lempel-Ziv decomposition of the string and finding palindromes. The book by Gusfield [18] devotes almost 150 pages to the applications of suffix trees, stressing the importance of these applications in bioinformatics. However, the massive increase of genomic data in the last decades requires space-efficient data structures able to efficiently support pattern matching queries, and the space consumption of suffix trees is too high. In 1990, Manber and Myers invented *suffix arrays* [25] as a space-efficient alternative to suffix trees. While suffix arrays do not have the full functionality of suffix trees, they still allow solving pattern matching queries. Suffix arrays started a new chapter in data compression, which culminated in the invention of data structures closely related to suffix arrays, notably, the Burrows-Wheeler Transform [4] and the FM-index [14, 16], which have heavily influenced sequence assembly [30].

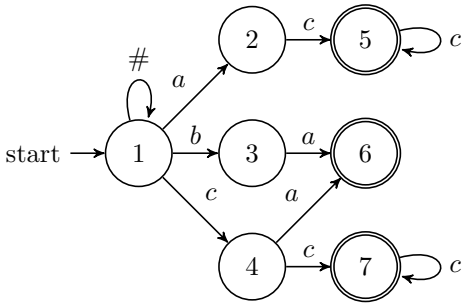
The impact of suffix arrays has led to a big effort in the attempt of designing efficient algorithms to construct suffix arrays, where "efficient" refers to various metrics (worst-case running time, average running time, space, performance on real data and so on); see [28] for a comprehensive survey on the topic. Let us focus on worst-case running time. Manber and Myers build the suffix array of a string of length n in $O(n \log n)$ by means of a *prefix-doubling* algorithm [25]. In 1997, Farach proposed a recursive algorithm to build the suffix *tree* of a

string in linear time for integer alphabets [12]. In the following years, the recursive paradigm of Farach's algorithm was used to developed a multitude of linear-time algorithms for building the suffix array [23, 20, 21, 26]. All these algorithms carefully exploit the lexicographic struture of the suffixes of a string, recursively reducing the problem of computing the suffix array of a string to the problem of computing the suffix array of a smaller string (*induced sorting*).

The problem of solving pattern matching queries not only on strings, but also on labeled graphs, is an active topic of research. Recently, Equi et al. showed that no algorithm can solve pattern matching queries *on arbitrary graphs* in $O(m^{1-\epsilon}|P|)$ time or $O(m|P|^{1-\epsilon})$ (where m is the number of edges, P is the pattern and $\epsilon > 0$), unless the Orthogonal Vectors hypothesis fails [11, 10]. On the other hand, over the years the idea of (lexicographically) sorting the suffixes of a string has been generalized to graphs, thus leading to compact data structures that are able to support pattern matching queries on graphs. The mechanism behind the suffix array, the Burrows-Wheeler Transform and the FM-index was first generalized to trees [13, 15]; later on, it was generalized to De Brujin graphs [3, 24] (which can be used for Eulerian sequence assembly [19]). Subsequently, it was extended to the so-called Wheeler graphs [17, 1], and finally to arbitrary graphs and automata [9, 6]. The idea of lexicographically sorting the strings reaching the states of an automaton has also deep theoretical consequences in automata theory: for example, it leads to a parametrization of the powerset construction, which implies fixed-parameter tractable algorithms for PSPACE-complete problems such as deciding the equivalence of two non-deterministic finite automata (NFAs) [8].

The case of deterministic finite automata (DFAs) is of particular interest, because in this case the notion of "suffix array" of a DFA has a simple interpretation in terms of strings. Assume that there is a fixed total order $\preceq$ on the alphabet Σ of a DFA $\mathcal{A}$, and extend $\preceq$ lexicographically to the set of all infinite strings on Σ. Assume that each state has at least one incoming edge. If u is a state, consider the set I_u of all infinite strings that can be read starting from u and following edges *in a backward fashion*, and let $\min_u$ and $\max_u$ be the lexicograpically smallest (largest, respectively) string in I_u (see Figure 1). Consider the partial order $\preceq_{\mathcal{A}}$ on the set of all states Q such that for every $u, v \in Q$, with $u \neq v$, it holds $u \prec_{\mathcal{A}} v$ if and only if $\max_u \preceq \min_v$. Then, the partial order $\preceq_{\mathcal{A}}$ induces a (partial) permutation of the set of all states that plays the same role of the permutation of text positions induced by the suffix array of a string [9, 22], so $\preceq_{\mathcal{A}}$ is essentially the "suffix array" of the DFA. If we are able to compute the partial order $\preceq_{\mathcal{A}}$ (and a minimum-size partition of Q into sets such that the restriction of $\preceq_{\mathcal{A}}$ to each set is a *total* order), then we can efficiently and compactly solve pattern matching queries on the DFA by means of techniques that extend the Burrows-Wheeler transform and the FM-index from strings to graphs. As a consequence, we now have to solve the problem of efficiently building the "suffix array" of a DFA, that is, the problem of computing $\preceq_{\mathcal{A}}$.

The first paper on the topic [9] presents an algorithm that builds $\preceq_{\mathcal{A}}$ in $O(m^2+n^{5/2})$ time, where n is the number of states and m is the number of edges. In a recent paper [22], Kim et al. describe two algorithms running in $O(mn)$ and $O(n^2 \log n)$ time. The key observation is that, if we build the *min/max-partition* of the set of all states, then we can determine the partial order $\preceq_{\mathcal{A}}$ in linear time by means of a reduction to the interval partitioning problem. Determining the min/max-partition of the set of all states means picking the string $\min_u$ and $\max_u$ for every state $u \in Q$, and sorting these $2|Q|$ strings (note that some of these strings may be equal, see Figure 1 for an example). As a consequence, we are only left with the problem of determining the min/max-partition efficiently.



i	$\min_i$	$\max_i$
1	$\#\#\#\#\#\ldots$	$\#\#\#\#\#\ldots$
2	$a\#\#\#\#\ldots$	$a\#\#\#\#\ldots$
3	$b\#\#\#\#\ldots$	$b\#\#\#\#\ldots$
4	$c\#\#\#\#\ldots$	$c\#\#\#\#\ldots$
5	$ca\#\#\#\ldots$	$ccccc\ldots$
6	$ab\#\#\#\ldots$	$ac\#\#\#\ldots$
7	$cc\#\#\#\ldots$	$ccccc\ldots$

Figure 1 A DFA $\mathcal{A}$, with the minimum and maximum string reaching each state (we assume $\# \prec a \prec b \prec c$). The min/max partition is given by $\{(1, \min), (1, \max)\} < \{(2, \min), (2, \max)\} < \{(6, \min)\} < \{(6, \max)\} < \{(3, \min), (3, \max)\} < \{(4, \min), (4, \max)\} < \{(5, \min)\} < \{(7, \min)\} < \{(5, \max), (7, \max)\}$, meaning that $\min_1 = \max_1 \prec \min_2 = \max_2 \prec \min_6 \prec \max_6 \prec \min_3 = \max_3 \prec \min_4 = \max_4 \prec \min_5 \prec \min_7 \prec \max_5 = \max_7$.

The $O(n^2 \log n)$ algorithm builds the min/max-partition by generalizing Manber and Myers's $O(n \log n)$ algorithm from strings to DFAs. However, since it is possible to build the suffix array of a string in $O(n)$ time, it is natural to wonder whether it is possible to determine the min/max-partition in $O(n^2)$ time.

In this paper, we show that, indeed, it is possible to build the min/max-partition in $O(n^2)$ time by adopting a recursive approach inspired by one of the linear-time algorithms that we have mentioned earlier, namely, Ko and Aluru's algorithm [23]. As a consequence, our algorithm is asymptotically faster than all previous algorithms.

A long-standing open problem is whether it is possible to define a suffix tree of a graph. Some recent work [5] suggests that it is possible to define data structures that *simulate* the behavior of a suffix tree by carefully studying the lexicographic structure of the graph (the results in [5] only hold for Wheeler graphs, but we believe that they can be extended to arbitrary graphs). More specifically, it is reasonable to believe that it is possible to generalize the notion of *compressed suffix tree* of a string [29] to graphs. A compressed suffix tree is a compressed representation of a suffix tree which consists of some components, including a suffix array. We have already seen that $\preceq_{\mathcal{A}}$ generalizes the suffix array to a graph structure, and [5] suggests that the remaining components may also be generalized to graphs. The complements of the suffix tree of a string heavily exploit the lexicographic and combinatorial structure of a string. Since the algorithm that we present in this paper deeply relies on the richness of the lexicographic structure (which becomes even more challenging and surprising when switching from a string setting to a graph setting), we believe that our results also provide a solid conceptual contribution towards extending suffix tree functionality to graphs.

We remark that, a few days after we submitted this paper to arXiv, a new arXiv preprint showed how to determine the min/max partition in $O(m \log n)$ time, where n is the number of states and m is the number of edges (this new arXiv preprint was also accepted for publication [2]). If the graph underlying the DFA is sparse, then the algorithm in [2] improves our $O(n^2)$ algorithm. Since the $O(m \log n)$ algorithm uses different techniques (it is obtained by adapting Paige and Tarjan's partition refinement algorithm [27]), we are left with the intriguing open problem of determining whether, by possibly combining the ideas behind our algorithm and the algorithm in [2], it is possible to build the min/max partition in $O(m)$ time.

Due to space constraints, all proofs can be found in the full version of this paper [7].

2 Preliminaries

2.1 Relation with Previous Work

In the setting of the previous works on the topic [1, 9, 22], the problem that we want to solve is defined as follows. Consider a deterministic finite automaton (DFA) such that (i) all edges entering the same state have the same label, (ii) each state is reachable from the initial state, (iii) each state is co-reachable, that is, it is either final or it allows reaching a final state, (iv) the initial state has no incoming edges. Then, determine the min/max partition of the set of states (see Section 2.2 for the formal definition of min/max partition, and see Figure 1 for an example).

- Assumptions (ii) and (iii) are standard assumptions in automata theory, because all states that do not satisfy these assumptions can be removed without changing the accepted language.
- In this setting, all the non-initial states have an incoming edge, but the initial state has no incoming edges. This implies for some state u it may hold $I_u = \emptyset$ (remember that I_u is the set of all infinite strings that can be read starting from u and following edges in a backward fashion, see the introduction), so Kim et al. [22] need to perform a tedious case analysis which also takes finite strings into account in order to define the min/max-partition (in particular, the minimum and maximum strings reaching the initial state are both equal to the empty string). However, we can easily avoid this complication by means of the same trick used in [5]; we can add a self-loop to the initial state, and the label of the self-loop is a special character # *smaller* than any character in the alphabet. Intuitively, # plays the same role as the termination character in the Burrows-Wheeler transform of a string, and since # is the smallest character, adding this self-loop does not affect the min/max-partition (see [22] for details).
- Notice that the initial state and the set of all final states play no role in the definition of the min/max partition; this explains why, more generally, it will be expedient to consider deterministic graphs rather than DFAs (otherwise we would need to artificially add an initial state and add a set of final states when we recursively build a graph in our algorithm). Equivalently, one may assume to work with semiautomata in which the transition function is not necessarily total. This justifies the assumptions that we will make in Section 2.2.
- Some recent papers [6, 8] have shown that assumptions (i) and (iv) can be removed. The partial order $\preceq_{\mathcal{A}}$ is defined analogously, and all the algorithms for building $\preceq_{\mathcal{A}}$ that we have mentioned still work. Indeed, if a state u is reached by edges with the distinct labels, we need to only consider all edges with the smallest label when computing $\min_u$ and all edges with the largest label when computing $\max_u$; moreover, we once again assume that the initial state has a self loop labeled #. The only difference is that assumption (i) implies that $m \leq n^2$ (n being the number of states, m being the number of edges) because each state can have at most n incoming edges, but this is no longer true if we remove assumption (i). As a consequence, the running time of our algorithm is no longer $O(n^2)$ but $O(m + n^2)$ (and the running time of the $O(n^2 \log n)$ algorithm in [22] becomes $O(m + n^2 \log n)$) because we still need to process all edges in the DFA.

To sum up, all the algorithms for computing $\preceq_{\mathcal{A}}$ work on arbitrary DFAs.

2.2 Notation and First Definitions

Let Σ be a finite alphabet. We consider finite, edge-labeled graphs $G = (V, E)$, where V is the set of all nodes and $E \subseteq V \times V \times \Sigma$ is the set of all edges. Up to taking a subset of Σ, we assume that all $c \in \Sigma$ label some edge in the graph. We assume that all nodes have at least one incoming edge, and all edges entering the same node u have the same label $\lambda(u)$ (*input consistency*). This implies that an edge $(u, v, a) \in E$ can be simply denoted as (u, v), because it must be $a = \lambda(v)$. In particular, it must be $|E| \leq |V|^2$ (and so an $O(|E|)$ algorithm is also a $O(|V|^2)$ algorithm). If we do not know the $\lambda(u)$'s, we can easily compute them by scanning all edges. In addition, we always assume that G is *deterministic*, that is, for every $u \in V$ and for every $a \in \Sigma$ there exists at most one $v \in V$ such that $(u, v) \in E$ and $\lambda(v) = a$.

Let Σ^* be the set of all finite strings on Σ, and let Σ^ω be the set of all (countably) right-infinite strings on Σ. If $\alpha \in \Sigma^* \cup \Sigma^\omega$ and $i \geq 1$, we denote by $\alpha[i] \in \Sigma$ the i^{th} character of α (that is, $\alpha = \alpha[1]\alpha[2]\alpha[3]\dots$). If $1 \leq i \leq j$, we define $\alpha[i, j] = \alpha[i]\alpha[i+1]\dots\alpha[j-1]\alpha[j]$, and if $j < i$, then $\alpha[i, j]$ is the empty string ϵ. If $\alpha \in \Sigma^*$, then $|\alpha|$ is length of α; for every $0 \leq j \leq |\alpha|$ the string $\alpha[1, j]$ is a *prefix* of α, and if $0 \leq j < |\alpha|$ it is a *strict prefix* of α; analogously, one defines suffixes and strict suffixes of α. An *occurrence* of $\alpha \in \Sigma^*$ starting at $u \in V$ and ending at $u' \in V$ is a sequence of nodes $u_1, u_2, \dots, u_{|\alpha|+1}$ of V such that (i) $u_1 = u$, (ii) $u_{|\alpha|+1} = u'$, (iii) $(u_{i+1}, u_i) \in E$ for every $1 \leq i \leq |\alpha|$ and (iv) $\lambda(u_i) = \alpha[i]$ for every $1 \leq i \leq |\alpha|$. An *occurrence* of $\alpha \in \Sigma^\omega$ starting at $u \in V$ is a sequence of nodes $(u_i)_{i \geq 1}$ of V such that (i) $u_1 = u$, (ii) $(u_{i+1}, u_i) \in E$ for every $i \geq 1$ and (iii) $\lambda(u_i) = \alpha[i]$ for every $i \geq 1$. Intuitively, a string $\alpha \in \Sigma^* \cup \Sigma^\omega$ has an occurrence starting at $u \in V$ if we can read α on the graph starting from u and following edges *in a backward fashion*.

In the paper, occurrences of strings in Σ^ω will play a key role, while occurrences of strings in Σ^* will be used as a technical tool. For every $u \in V$, we denote by I_u the set of all strings in Σ^ω admitting an occurrence starting at u. Since every node has at least one incoming edge, then $I_u \neq \emptyset$.

A *total order* $\leq$ on a set V if a reflexive, antisymmetric and transitive relation on V. If $u, v \in V$, we write $u < v$ if $u \leq v$ and $u \neq v$.

Let $\preceq$ be a fixed total order on Σ. We extend $\preceq$ to $\Sigma^* \cup \Sigma^\omega$ *lexicographically*. It is easy to show that in every I_u there is a lexicographically smallest string $\min_u$ and a lexicographically largest string $\max_u$ (for example, it follows from [22, Observation 8]).

We will often use the following immediate observation. Let $u \in V$, and let $(u_i)_{i \geq 1}$ be an occurrence of $\min_u$. Fix $i \geq 1$. Then, $(u_j)_{j \geq i}$ is an occurrence of $\min_{u_i}$, and $\min_u = \min_u[1, i-1]\min_{u_i}$.

Let $V' \subseteq V$. Let $\mathcal{A}$ be the unique partition of V' and let $\leq$ be the unique total order on $\mathcal{A}$ such that, for every $I, J \in \mathcal{A}$ and for every $u \in I$ and $v \in J$, (i) if $I = J$, then $\min_u = \min_v$ and (ii) if $I < J$, then $\min_u \prec \min_v$. Then, we say that $(\mathcal{A}, \leq)$, or more simply $\mathcal{A}$, is the *min-partition* of V'. The *max-partition* of V' is defined analogously. Now, consider the set $V' \times \{\min, \max\}$, and define $\rho((u, \min)) = \min_u$ and $\rho((u, \max)) = \max_u$ for every $u \in V'$. Let $\mathcal{B}$ be the unique partition of $V' \times \{\min, \max\}$ and let $\leq$ be the unique total order on $\mathcal{B}$ such that, for every $I, J \in \mathcal{B}$ and for every $x \in I$ and $y \in J$, (i) if $I = J$, then $\rho(x) = \rho(y)$ and (ii) if $I < J$, then $\rho(x) \prec \rho(y)$. Then, we say that $(\mathcal{B}, \leq)$, or more simply $\mathcal{B}$, is the *min/max-partition* of V'.

The main result of this paper will be proving that the min/max partition of V can be determined in $O(|V|^2)$ time.

2.3 Our Approach

Let $G = (V, E)$ be a graph. We will first show how to build the min-partition of V in $O(n^2)$ time, where $n = |V|$ (Section 4); then, we will show how the algorithm can be adapted so that it builds the min/max-partition in $O(n^2)$ time (Section 5).

In order to build a min-partition of V, we will first classify all minima into three categories (Section 3), so that we can split V into three pairwise-disjoint sets V_1, V_2, V_3. Then, we will show that in $O(n^2)$ time:

- we can compute V_1, V_2, V_3 (Section 4.1);
- we can define a graph $\bar{G} = (\bar{V}, \bar{E})$ having $|V_3|$ nodes (Section 4.2);
- assuming that we have already determined the min-partition of $\bar{V}$, we can determine the min-partition of V (Section 4.3).

Analogously, in $O(n^2)$ time we can reduce the problem of determining the min-partition of V to the problem of determining the min-partition of the set of all nodes of a graph having $|V_1|$ (not $|V_3|$) nodes (Section 4.4). As a consequence, since $\min\{|V_1|, |V_3|\} \leq |V|/2 = n/2$, we obtain a recursive algorithm whose running time is given by the recurrence:

$$T(n) = T(n/2) + O(n^2)$$

and we conclude that the running time of our algorithm is $O(n^2)$.

3 Classifying Strings

In [23], Ko and Aluru divide the suffixes of a string into two groups. Here we follow an approach purely based on stringology, without fixing a string or a graph from the start. We divide the strings of Σ^ω into three groups, which we call group 1, group 2 and group 3 (Corollary 3 provides the intuition behind this choice).

▶ **Definition 1.** *Let $\alpha \in \Sigma^\omega$. Let $a \in \Sigma$ and $\alpha' \in \Sigma^\omega$ such that $\alpha = a\alpha'$. Then, we define $\tau(\alpha)$ as follows:*

1. *$\tau(\alpha) = 1$ if $\alpha' \prec \alpha$.*
2. *$\tau(\alpha) = 2$ if $\alpha' = \alpha$.*
3. *$\tau(\alpha) = 3$ if $\alpha \prec \alpha'$.*

We will constantly use the following characterization.

▶ **Lemma 2.** *Let $\alpha \in \Sigma^\omega$. Let $a \in \Sigma$ and $\alpha' \in \Sigma^\omega$ such that $\alpha = a\alpha'$. Then:*

1. *$\tau(\alpha) = 2$ if and only if $\alpha' = a^\omega$, if and only if $\alpha = a^\omega$.*
2. *$\tau(\alpha) \neq 2$ if and only if $\alpha' \neq a^\omega$, if and only if $\alpha \neq a^\omega$.*

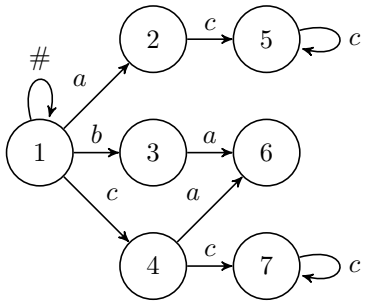
Assume that $\tau(\alpha) \neq 2$. Then, there exist unique $c \in \Sigma \setminus \{a\}$, $\alpha'' \in \Sigma^\omega$ and $i \geq 0$ such that $\alpha' = a^i c\alpha''$ (and so $\alpha = a^{i+1}c\alpha''$). Moreover:

1. *$\tau(\alpha) = 1$ if and only if $c \prec a$, if and only if $\alpha' \prec a^\omega$, if and only if $\alpha \prec a^\omega$.*
2. *$\tau(\alpha) = 3$ if and only if $a \prec c$, if and only if $a^\omega \prec \alpha'$, if and only if $a^\omega \prec \alpha$,*

The following corollary will be a key ingredient in our recursive approach.

▶ **Corollary 3.** *Let $\alpha, \beta \in \Sigma^\omega$. Let $a, b \in \Sigma$ and $\alpha', \beta' \in \Sigma^\omega$ such that $\alpha = a\alpha'$ and $\beta = b\beta'$. Then:*

1. *If $a = b$ and $\tau(\alpha) = \tau(\beta) = 2$, then $\alpha = \beta$.*
2. *If $a = b$ and $\tau(\alpha) < \tau(\beta)$, then $\alpha \prec \beta$. Equivalently, if $a = b$ and $\alpha \preceq \beta$, then $\tau(\alpha) \leq \tau(\beta)$.*



i	$\min_i$	$\tau(i)$ $(= \tau(\min_i))$
1	$\#\#\#\#\#\ldots$	2
2	$a\#\#\#\#\ldots$	1
3	$b\#\#\#\#\ldots$	1
4	$c\#\#\#\#\ldots$	1
5	$ca\#\#\#\ldots$	1
6	$ab\#\#\#\ldots$	3
7	$cc\#\#\#\ldots$	1

Figure 2 The graph from Figure 1, with the values $\min_i$'s and $\tau(i)$'s.

4 Computing the min-partition

Let $G = (V, E)$ be a graph. We will prove that we can compute the min-partition of V in $O(|V|^2)$ time. In this section, for every $u \in V$ we define $\tau(u) = \tau(\min_u)$ (see Figure 2).

Let $u \in V$, and let $(u_i)_{i \geq 1}$ be an occurrence of $\min_u$ starting at u. It is immediate to realize that (i) if $\tau(u) = 1$, then $\lambda(u_2) \preceq \lambda(u_1)$, (ii) if $\tau(u) = 2$, then $\lambda(u_k) = \lambda(u_1)$ for every $k \geq 1$ and (iii) if $\tau(u) = 3$, then $\lambda(u_1) \preceq \lambda(u_2)$.

As a first step, let us prove that without loss of generality we can remove some edges from G without affecting the min/max-partition. This preprocessing will be helpful in Lemma 23.

▶ **Definition 4.** *Let $G = (V, E)$ be a graph. We say that G is* trimmed *if it contains no edge $(u, v) \in E$ such that $\tau(v) = 1$ and $\lambda(v) \prec \lambda(u)$.*

In order to simplify the readability of our proofs, we will not directly remove some edges from $G = (V, E)$, but we will first build a copy of G where every node u is a mapped to a node u^*, and then we will trim the graph. In this way, when we write $\min_u$ and $\min_{u^*}$ it will be always clear whether we refer to the original graph or the trimmed graph. We will use the same convention in Section 4.2 when we define the graph $\bar{G} = (\bar{V}, \bar{E})$ that we will use for the recursive step.

▶ **Lemma 5.** *Let $G = (V, E)$ be a graph. Then, in $O(|E|)$ time we can build a trimmed graph $G^* = (V^*, E^*)$, with $V^* = \{u^* \mid u \in V\}$, such that for every $u \in V$ it holds $\min_{u^*} = \min_u$. In particular, $\tau(u^*) = \tau(u)$ for every $u \in V$.*

4.1 Classifying Minima

Let us first show how to compute all $u \in V$ such that $\tau(u) = 1$.

▶ **Lemma 6.** *Let $G = (V, E)$ be a graph, and let $u, v \in V$.*
1. *If $(u, v) \in E$ and $\lambda(u) \prec \lambda(v)$, then $\tau(v) = 1$.*
2. *If $(u, v) \in E$, $\lambda(u) = \lambda(v)$ and $\tau(u) = 1$, then $\tau(v) = 1$.*

▶ **Corollary 7.** *Let $G = (V, E)$ be a graph, and let $u \in V$. Then, $\tau(u) = 1$ if and only if there exist $k \geq 2$ and $z_1, \ldots, z_k \in V$ such that (i) $(z_i, z_{i+1}) \in E$ for every $1 \leq i \leq k - 1$, (ii) $z_k = u$, (iii) $\lambda(z_1) \prec \lambda(z_2)$ and (iv) $\lambda(z_2) = \lambda(z_3) = \cdots = \lambda(z_k)$.*

Corollary 7 yields an algorithm to decide whether $u \in V$ is such that $\tau(u) = 1$.

▶ **Corollary 8.** *Let $G = (V, E)$ be a graph. We can determine all $u \in V$ such that $\tau(u) = 1$ in time $O(|E|)$.*

Now, let us show how to determine all $u \in V$ such that $\tau(u) = 2$. We can assume that we have already determined all $u \in V$ such that $\tau(u) = 1$.

▶ **Lemma 9.** *Let $G = (V, E)$ be a graph, and let $u \in V$ such that $\tau(u) \neq 1$. Then, we have $\tau(u) = 2$ if and only if there exist $k \geq 2$ and $z_1, \ldots, z_k \in V$ such that (i) $(z_{i+1}, z_i) \in E$ for every $1 \leq i \leq k-1$, (ii) $z_1 = u$, (iii) $z_k = z_j$ for some $1 \leq j \leq k-1$ and (iv) $\lambda(z_1) = \lambda(z_2) = \cdots = \lambda(z_k)$.*

In particular, such $z_1, \ldots, z_k \in V$ must satisfy $\tau(z_i) = 2$ for every $1 \leq i \leq k$.

▶ **Corollary 10.** *Let $G = (V, E)$ be a graph. We can determine all $u \in V$ such that $\tau(u) = 2$ in time $O(|E|)$.*

From Corollary 8 and Corollary 10 we immediately obtain the following result.

▶ **Corollary 11.** *Let $G = (V, E)$ be a graph. Then, in time $O(|E|)$ we can compute $\tau(u)$ for every $u \in V$.*

4.2 Recursive Step

Let us sketch the general idea to build a smaller graph for the recursive step. We consider each $u \in V$ such that $\tau(u) = 3$, and we follow edges in a backward fashion, aiming to determine a prefix of $\min_u$. As a consequence, we discard edges through which no occurrence of $\min_u$ can go, and by Corollary 3 we can restrict our attention to the nodes v such that $\tau(v)$ is minimal. We proceed like this until we encounter nodes v' such that $\tau(v') = 3$.

Let us formalize our intuition. We will first present some properties that the occurrences of a string $\min_u$ must satisfy.

▶ **Lemma 12.** *Let $G = (V, E)$ be a graph. Let $u, v \in V$ be such that $\min_u = \min_v$. Let $(u_i)_{i \geq 1}$ be an occurrence of $\min_u$ and let $(v_i)_{i \geq 1}$ be an occurrence of $\min_v$. Then:*

1. *$\lambda(u_i) = \lambda(v_i)$ for every $i \geq 1$.*
2. *$\min_{u_i} = \min_{v_i}$ for every $i \geq 1$.*
3. *$\tau(u_i) = \tau(v_i)$ for every $i \geq 1$.*

In particular, the previous results hold if $u = v$ and $(u_i)_{i \geq 1}$ and $(v_i)_{i \geq 1}$ are two distinct occurrences of $\min_u$.

▶ **Lemma 13.** *Let $G = (V, E)$ be a graph. Let $u \in V$ and let $(u_i)_{i \geq 1}$ an occurrence of $\min_u$ starting at u. Let $k \geq 1$ be such that $\tau(u_1) = \tau(u_2) = \cdots = \tau(u_{k-1}) = \tau(u_k) \neq 2$. Then, $u_1, \ldots, u_k$ are pairwise distinct. In particular, $k \leq |V|$.*

The previous results allow us to give the following definition.

▶ **Definition 14.** *Let $G = (V, E)$ be a graph. Let $u \in V$ such that $\tau(u) = 3$. Let ℓ_u to be the smallest integer $k \geq 2$ such that $\tau(u_k) \geq 2$, where $(u_i)_{i \geq 1}$ is an occurrence of $\min_u$ starting at u.*

Note that ℓ_u is well-defined, because (i) it cannot hold $\tau(u_k) = 1$ for every $k \geq 2$ by Lemma 13 (indeed, if $\tau(u_2) = 1$, then $(u_i)_{i \geq 2}$ is an occurrence of $\min_{u_2}$ starting at u_2, and by Lemma 13 there exists $2 \leq k \leq |V| + 2$ such that $\tau(u_k) \neq 1$) and (ii) ℓ_u does not depend on the choice of $(u_i)_{i \geq 1}$ by Lemma 12. In particular, it must be $\ell_u \leq |V| + 1$ because $u_1, u_2, \ldots, u_{\ell_u - 1}$ are pairwise distinct (u_1 is distinct from $u_2, \ldots, u_{\ell_u - 1}$ because $\tau(u_1) = 3$ and $\tau(u_2) = \tau(u_3) = \ldots \tau(u_{\ell_u - 1}) = 1$ by the minimality of ℓ_u).

▶ **Lemma 15.** *Let $G = (V, E)$ be a graph. Let $u \in V$ such that $\tau(u) = 3$. Then, $\min_u[i+1] \preceq \min_u[i]$ for every $2 \leq i \leq \ell_u - 1$. In particular, if $2 \leq i \leq j \leq \ell_u$, then $\min_u[j] \preceq \min_u[i]$.*

If $R \subseteq Q$ is a nonempty set of nodes such that for every $u, v \in R$ it holds $\lambda(u) = \lambda(v)$, we define $\lambda(R) = \lambda(u) = \lambda(v)$. If $R \subseteq Q$ is a nonempty set of nodes such that for every $u, v \in R$ it holds $\tau(u) = \tau(v)$, we define $\tau(R) = \tau(u) = \tau(v)$.

Let $R \subseteq Q$ be a nonempty set of states. Let $\mathtt{F}(R) = \arg\min_{u \in R'} \tau(u)$, where $R' = \arg\min_{v \in R} \lambda(v)$. Notice that $\mathtt{F}(R)$ is nonempty, and both $\lambda(\mathtt{F}(R))$ and $\tau(\mathtt{F}(R))$ are well-defined. In other words, $\mathtt{F}(R)$ is obtained by first considering the subset $R' \subseteq \mathtt{F}(R)$ of all nodes v such that $\lambda(v)$ is as small as possible, and then considering the subset of R' of all nodes v such that $\tau(v)$ is as small as possible. This is consistent with our intuition on how we should be looking for a prefix of $\min_u$.

Define:

$$G_i(u) = \begin{cases} \{u\} & \text{if } i = 1; \\ \mathtt{F}(\{v' \in Q \mid (\exists v \in G_{i-1}(u))((v', v) \in E)\}) \setminus \bigcup_{j=2}^{i-1} G_j(u) & \text{if } 1 < i \leq \ell_u. \end{cases}$$

Notice that we also require that a node in $G_i(u)$ has not been encountered before. Intuitively, this does not affect our search for a prefix of $\min_u$ because, if we met the same node twice, then we would have a cycle where all edges are equally labeled (because by Lemma 15 labels can only decrease), and since $\tau(G_i(u)) = 1$ for every $2 \leq i \leq \ell_u - 1$, then no occurrence of the minimum can go through the cycle because if we remove the cycle from the occurrence we obtain a smaller string by Lemma 2.

The following technical lemma is crucial to prove that our intuition is correct.

▶ **Lemma 16.** *Let $G = (V, E)$ be a graph. Let $u \in V$ such that $\tau(u) = 3$.*

1. *$G_i(u)$ is well-defined and nonempty for every $1 \leq i \leq \ell_u$.*
2. *Let $(u_i)_{i \geq 1}$ be an occurrence of $\min_u$ starting at u. Then, $u_i \in G_i(u)$ for every $1 \leq i \leq \ell_u$. In particular, $\tau(u_i) = \tau(G_i(u))$ and $\min_u[i] = \lambda(u_i) = \lambda(G_i(u))$ for every $1 \leq i \leq \ell_u$.*
3. *For every $1 \leq i \leq \ell_u$ and for every $v \in G_i(u)$ there exists an occurrence of $\min_u[1, i-1]$ starting at u and ending at v.*

Let $u \in V$ such that $\tau(u) = 3$. We define:

- $\gamma_u = \min_u[1, \ell_u]$;
- $\mathtt{t}_u = \tau(G_{\ell_u}(u)) \in \{2, 3\}$

Now, in order to define the smaller graph for the recursive step, we also need a new alphabet $(\Sigma', \preceq')$, which must be defined consistently with the mutual ordering of the minima. The next lemma yields all the information that we need.

▶ **Lemma 17.** *Let $G = (V, E)$ be a graph. Let $u, v \in V$ such that $\tau(u) = \tau(v) = 3$. Assume that one of the following statements is true:*

1. *γ_u is not a prefix of γ_v and $\gamma_u \prec \gamma_v$.*
2. *$\gamma_u = \gamma_v$, $\mathtt{t}_u = 2$ and $\mathtt{t}_v = 3$.*
3. *γ_v is a strict prefix of γ_u.*

Then, $\min_u \prec \min_v$.

Equivalently, if $\min_u \preceq \min_v$, then one the following is true: (i) γ_u is not a prefix of γ_v and $\gamma_u \prec \gamma_v$; (ii) $\gamma_u = \gamma_v$ and $\mathtt{t}_u \leq \mathtt{t}_v$; (iii) γ_v is a strict prefix of γ_u.

Now, let $\Sigma' = \{(\gamma_u, \mathtt{t}_u) \mid u \in V, \tau(u) = 3\}$, and let $\preceq'$ be the total order on Σ' such that for every distinct $(\alpha, x), (\beta, y) \in \Sigma'$, it holds $(\alpha, x) \prec' (\beta, y)$ if and only if one of the following is true:

1. α is not a prefix of β and $\alpha \prec \beta$.
2. $\alpha = \beta$, $x = 2$ and $y = 3$.
3. β is a strict prefix of α.

It is immediate to verify that $\preceq'$ is a total order: indeed, $\preceq'$ is obtained (i) by first comparing the γ_u's using the variant of the (total) lexicographic order on Σ^* in which a string is smaller than every strict prefix of it and (ii) if the γ_u's are equal by comparing the $\mathtt{t}_u$'s, which are elements in $\{2,3\}$.

Starting from $G=(V,E)$, we define a new graph $\bar{G}=(\bar{V},\bar{E})$ as follows:

- $\bar{V}=\{\bar{u} \mid u\in V, \tau(u)=3\}$.
- The new totally-ordered alphabet is $(\Sigma',\preceq')$.
- For every $\bar{u}\in\bar{V}$, we define $\lambda(\bar{u})=(\gamma_u,\mathtt{t}_u)$.
- $\bar{E}=\{(\bar{v},\bar{v}) \mid \mathtt{t}_v=2\}\cup\{(\bar{u},\bar{v}) \mid \mathtt{t}_v=3, u\in G_{\ell_v}(v)\}$.

Note that for every $\bar{v}\in\bar{V}$ such that $\mathtt{t}_v=3$ and for every $u\in G_{\ell_v}(v)$ it holds $\tau(u)=\tau(G_{\ell_v}(v))=\mathtt{t}_v=3$, so $\bar{u}\in\bar{V}$ and $(\bar{u},\bar{v})\in E$. Moreover, $\bar{G}=(\bar{V},\bar{E})$ satisfies all the assumptions about graphs that we use in this paper: (i) all edges entering the same node have the same label (by definition), (ii) every node has at least one incoming edge (because if $\bar{v}\in\bar{V}$, then $G_{\ell_v}(v)\neq\emptyset$ by Lemma 16) and (iii) $\bar{G}$ is deterministic (because if $(\bar{u},\bar{v}),(\bar{u},\bar{v}')\in\bar{E}$ and $\lambda(\bar{v})=\lambda(\bar{v}')$, then $\gamma_v=\gamma_{v'}$ and $\mathtt{t}_v=\mathtt{t}_{v'}$, so by the definition of $\bar{E}$ if $\mathtt{t}_v=\mathtt{t}_{v'}=2$ we immediately obtain $\bar{v}=\bar{u}=\bar{v}'$, and if $\mathtt{t}_v=\mathtt{t}'_v=3$ we obtain $u\in G_{\ell_v}(v)\cap G_{\ell_{v'}}(v')$; since by Lemma 16 there exist two occurrences of $\min_v[1,\ell_v-1]=\gamma_v[1,\ell_v-1]=\gamma_{v'}[1,\ell_{v'}-1]=\min_{v'}[1,\ell_{v'}-1]$ starting at v and v' and both ending at u, the determinism of G implies $v=v'$ and so $\bar{v}=\bar{v}'$).

Notice that if $\bar{v}\in\bar{V}$ is such that $\mathtt{t}_v=2$, then $I_{\bar{v}}$ contains exactly one string, namely, $\lambda(\bar{v})^\omega$; in particular, $\min_{\bar{v}}=\max_{\bar{v}}=\lambda(\bar{v})^\omega$.

When we implement $G=(V,E)$ and $\bar{G}=(\bar{V},\bar{E})$, we use integer alphabets $\Sigma=\{0,1,\ldots,|\Sigma|-1\}$ and $\Sigma'=\{0,1,\ldots,|\Sigma'|-1\}$; in particular, we will not store Σ' by means of pairs $(\gamma_u,\mathtt{t}_u)$'s, but we will remap Σ' to an integer alphabet consistently with the total order $\preceq'$ on Σ', so that the mutual order of the $\min_{\bar{u}}$'s is not affected.

Let us prove that we can use $\bar{G}=(\bar{V},\bar{E})$ for the recursive step. We will start with some preliminary results.

▶ **Lemma 18.** *Let $G=(V,E)$ be a graph. Let $u,v\in V$ be such that $\tau(u)=\tau(v)=3$, $\gamma_u=\gamma_v$ and $\mathtt{t}_u=\mathtt{t}_v=2$. Then, $\min_u=\min_v$.*

▶ **Lemma 19.** *Let $G=(V,E)$ be a graph. Let $u\in V$, and let $(u_i)_{i\geq 1}$ be an occurrence of $\min_u$ starting at u. Then, exactly one of the following holds true:*

1. *There exists $i_0\geq 1$ such that $\tau(u_i)\neq 2$ for every $1\leq i<i_0$ and $\tau(u_i)=2$ for every $i\geq i_0$.*
2. *$\tau(u_i)\neq 2$ for every $i\geq 1$, and both $\tau(u_i)=1$ and $\tau(u_i)=3$ are true for infinitely many i's.*

Crucially, the next lemma establishes a correspondence between minima of nodes in $G=(V,E)$ and minima of nodes in $\bar{G}=(\bar{V},\bar{E})$.

▶ **Lemma 20.** *Let $G=(V,E)$ be a graph. Let $u\in V$ such that $\tau(u)=3$. Let $(u_i)_{i\geq 1}$ be an occurrence of $\min_u$ starting at u. Let $(u'_i)_{i\geq 1}$ be the infinite sequence of nodes in V obtained as follows. Consider $L=\{k\geq 1 \mid \tau(u_k)=3\}$, and for every $i\geq 1$, let $j_i\geq 1$ be the i^{th} smallest element of L, if it exists. For every $i\geq 1$ such that j_i is defined, let $u'_i=u_{j_i}$, and if $i\geq 1$ is such that j_i is not defined (so L is a finite set), let $u'_i=u'_{|L|}$. Then, $(\bar{u}'_i)_{i\geq 1}$ is an occurrence of $\min_{\bar{u}}$ starting at $\bar{u}$ in $\bar{G}=(\bar{V},\bar{E})$.*

The following theorem shows that our reduction to $\bar{G}=(\bar{V},\bar{E})$ is correct.

▶ **Theorem 21.** *Let $G = (V, E)$ be a graph. Let $u, v \in V$ be such that $\tau(u) = \tau(v) = 3$.*
1. *If $\min_u = \min_v$, then $\min_{\bar{u}} = \min_{\bar{v}}$.*
2. *If $\min_u \prec \min_v$, then $\min_{\bar{u}} \prec' \min_{\bar{v}}$.*

Since $\preceq$ is a total order (so exactly one among $\min_u \prec \min_v$, $\min_u = \min_v$ and $\min_v \prec \min_u$ holds true), from Theorem 21 we immediately obtain the following result.

▶ **Corollary 22.** *Let $G = (V, E)$ be a graph. Let $u, v \in V$ be such that $\tau(u) = \tau(v) = 3$.*
1. *It holds $\min_u = \min_v$ if and only if $\min_{\bar{u}} = \min_{\bar{v}}$.*
2. *It holds $\min_u \prec \min_v$ if and only if $\min_{\bar{u}} \prec' \min_{\bar{v}}$.*

In particular, if we have the min-partition of $\bar{V}$ (with respect to $\bar{G}$), then we also have the min-partition of $\{u \in V \mid \tau(u) = 3\}$ (with respect to G).

Lastly, we show that our reduction to $\bar{G} = (\bar{V}, \bar{E})$ can be computed within $O(n^2)$ time.

▶ **Lemma 23.** *Let $G = (V, E)$ be a trimmed graph. Then, we can build $\bar{G} = (\bar{V}, \bar{E})$ in $O(|V|^2)$ time.*

4.3 Merging

We want to determine the min-partition $\mathcal{A}$ of V, assuming that we already have the min-partition $\mathcal{B}$ of $\{u \in V \mid \tau(u) = 3\}$.

First, note that we can easily build the min-partition $\mathcal{B}'$ of $\{u \in V \mid \tau(u) = 2\}$. Indeed, if $\tau(u) = 2$, then $\min_u = \lambda(u)^\omega$ by Lemma 2. As a consequence, if $\tau(u) = \tau(v) = 2$, then (i) $\min_u = \min_v$ if and only if $\lambda(u) = \lambda(v)$ and (ii) $\min_u \prec \min_v$ if and only if $\lambda(u) \prec \lambda(v)$, so we can build $\mathcal{B}'$ in $O(|V|)$ time by using counting sort.

For every $c \in \Sigma$ and $t \in \{1, 2, 3\}$, let $V_{c,t} = \{v \in V \mid \lambda(v) = c, \tau(v) = t\}$. Consider $u, v \in V$: (i) if $\lambda(u) \prec \lambda(v)$, then $\min_u \prec \min_v$ and (ii) if $\lambda(u) = \lambda(v)$ and $\tau(u) < \tau(v)$, then $\min_u \prec \min_v$ by Corollary 3. As a consequence, in order to build $\mathcal{A}$, we only have to build the min-partition $\mathcal{A}_{c,t}$ of $V_{c,t}$, for every $c \in \Sigma$ and every $t \in \{1, 2, 3\}$.

A possible way to implement each $\mathcal{A}_{c,t}$ is by means of an array $A_{c,t}$ storing the elements of $V_{c,t}$, where we also use a special character to delimit the border between consecutive elements of $A_{c,t}$.

It is immediate to build incrementally $\mathcal{A}_{c,3}$ for every $c \in \Sigma$, from its smallest element to its largest element. At the beginning, $A_{c,3}$ is empty for every $c \in \Sigma$. Then, scan the elements I in $\mathcal{B}$ from smallest to largest, and add I to $\mathcal{A}_{c,3}$, where $c = \lambda(u)$ for any $u \in I$ (the definition of c does not depend on the choice of u). We scan $\mathcal{B}$ only once, so this step takes $O(|V|)$ time. Analogously, we can build $\mathcal{A}_{c,2}$ for every $c \in \Sigma$ by using $\mathcal{B}'$.

We are only left with showing how to build $\mathcal{A}_{c,1}$ for every $c \in \Sigma$. At the beginning, each $A_{c,1}$ is empty, and we will build each $\mathcal{A}_{c,1}$ from its smallest element to its largest element. During this step of the algorithm, we will gradually mark the nodes $u \in V$ such that $\tau(u) = 1$. At the beginning of the step, no such node is marked, and at the end of the step all these nodes will be marked. Let $\Sigma = \{c_1, c_2, \ldots, c_\sigma\}$, with $c_1 \prec c_2 \prec \cdots \prec c_\sigma$. Notice that it must be $V_{c_1,1} = \emptyset$, because if there existed $u \in V_{c_1,1}$, then it would be $\min_u \prec c_1^\omega$ by Lemma 2 and so c_1 would not be the smallest character in Σ. Now, consider $V_{c_1,2}$; we have already fully computed $A_{c_1,2}$. Process each I in $A_{c_1,2}$ from smallest to largest, and for every $c_k \in \Sigma$ compute the set J_k of all non-marked nodes $v \in V$ such that $\tau(v) = 1$, $\lambda(v) = c_k$, and $(u, v) \in E$ for some $u \in I$. Then, if $J_k \neq \emptyset$ add J_k to $A_{c_k,1}$ and mark the nodes in J_k. After processing the elements in $A_{c_1,2}$, we process the element in $A_{c_1,3}$, $A_{c_2,1}$, $A_{c_2,2}$, $A_{c_2,3}$, $A_{c_3,1}$ and so on, in this order. Each $A_{c_i,t}$ is processed from its (current) smallest element to its (current) largest element. We never remove or modify elements in any $A_{c,t}$, but we only

add elements to the $A_{c,1}$'s. More precisely, when we process I in $A_{c,t}$, for every $c_k \in \Sigma$ we compute the set J_k of all non-marked nodes $v \in V$ such that $\tau(v) = 1$, $\lambda(v) = c_k$, and $(u, v) \in E$ for some $u \in I$ and, if $J_k \neq \emptyset$, then we add J_k to $A_{c_k,1}$ and we mark the nodes in J_k.

The following lemma shows that our approach is correct. Let us give some intuition. A *prefix* of a min-partition $\mathcal{C}$ is a subset $\mathcal{C}'$ of $\mathcal{C}$ such that, if $I, J \in \mathcal{C}$, $I < J$ and $J \in \mathcal{C}'$, then $I \in \mathcal{C}'$. Notice that every prefix of $\mathcal{A}$ is obtained by taking the union of $\mathcal{A}_{c_1,2}$, $\mathcal{A}_{c_1,3}$, $\mathcal{A}_{c_2,1}$, $\mathcal{A}_{c_2,2}$, $\mathcal{A}_{c_2,3}$, $\mathcal{A}_{c_3,1}$, ... in this order up to some element $\mathcal{A}_{c,t}$, where possibly we only pick a prefix of the last element $\mathcal{A}_{c,t}$. Then, we will show that, when we process I in $A_{c,t}$, we have already built the prefix of $\mathcal{A}$ whose largest element is I. This means that, for every $v \in J_k$ and for any *any* occurrence $(v_i)_{i\geq 1}$ of $\min_v$ starting at v, it must hold that v_2 is in I.

▶ **Lemma 24.** *Let $G = (V, E)$ be a graph. If we know the min-partition of $\{u \in V \mid \tau(u) = 3\}$, then we can build the min-partition of V in $O(|E|)$ time.*

4.4 The Complementary Case

We have shown that in $O(n^2)$ time we can reduce the problem of determining the min-partition of V to the problem of determining the min-partition of the set of all nodes of a graph having $|\{u \in V \mid \tau(u) = 3\}|$ nodes. Now, we must show that (similarly) in $O(n^2)$ time we can reduce the problem of determining the min-partition of V to the problem of determining the min-partition of the set of all nodes of a graph having $|\{u \in V \mid \tau(u) = 1\}|$ nodes. The merging step will be more complex, because the order in which we will process the $\mathcal{A}_{c,t}$ will be from largest to smallest ($\mathcal{A}_{c_\sigma,2}$, $\mathcal{A}_{c_\sigma,1}$, $\mathcal{A}_{c_{\sigma-1},3}$, $\mathcal{A}_{c_{\sigma-1},2}$, $\mathcal{A}_{c_{\sigma-1},1}$, $\mathcal{A}_{c_{\sigma-2},3}$ and so on) so we will need to update some elements of some $A_{c,t}$'s to include the information about minima that we may infer at a later stage of the algorithm. We provide the details in the full version of the paper [7].

5 Computing the min/max-partition

Let $G = (V, E)$ be a graph. We can build the *max-partition* of V by simply considering the transpose total order $\preceq^*$ of $\preceq$ (the one for which $a \preceq^* b$ if and only if $b \preceq a$) and building the min-partition. As a consequence, the algorithm to build the max-partition is entirely symmetrical to the algorithm to build the min-partition.

Let $G = (V, E)$ be a graph. Let us show how we can build the min/max-partition of V in $O(|V|^2)$ time. Assume that we have two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ on the same alphabet $(\Sigma, \preceq)$, with $V_1 \cap V_2 = \emptyset$ (we allow G_1 and G_2 to possibly be the *null graph*, that is, the graph without vertices). Let $V_1' \subseteq V_1$, $V_2' \subseteq V_2$, $W = V_1' \cup V_2'$, and for every $u \in W$ define $\rho(u) = \min_u$ if $u \in V_1'$, and $\rho(u) = \max_u$ if $u \in V_2'$. Let $\mathcal{A}$ be the unique partition of W and let $\leq$ be the unique total order on $\mathcal{A}$ such that, for every $I, J \in \mathcal{A}$ and for every $u \in I$ and $u \in J$, (i) if $I = J$, then $\rho(u) = \rho(u)$ and (ii) if $I < J$, then $\rho(u) \prec \rho(u)$. Then, we say that $(\mathcal{A}, \leq)$, or more simply $\mathcal{A}$, is the *min/max-partition* of (V_1', V_2'). We will show that we can compute the min/max partition of (V_1, V_2) in $O((|V_1| + |V_2|)^2)$ time. In particular, if $G_1 = (V_1, E_1)$ and $G_2 = (V_1, E_2)$ are two (distinct) copies of the same graph $G = (V, E)$, then we can compute the min/max-partition of V in $O(|V|^2)$ time.

We compute $\tau(\min_u)$ for every $u \in V_1$ and we compute $\tau(\max_u)$ for every $u \in V_2$. If the number of values equal to 3 is smaller than the number of values equal to 1, then (in time $O(|V_1|^2 + |V_2|^2) = O((|V_1| + |V_2|)^2)$) we build the graphs $\bar{G}_1 = (\bar{V}_1, \bar{E}_1)$ and $\bar{G}_2 = (\bar{V}_2, \bar{E}_2)$ as defined before, where $\bar{V}_1 = \{\bar{u} \mid u \in V_1, \tau(\min_u) = 3\}$ and $\bar{V}_2 = \{\bar{u} \mid u \in V_2, \tau(\max_u) = 3\}$, otherwise we consider the complementary case (which is symmetrical). When building $\bar{G}_1 = (\bar{V}_1, \bar{E}_1)$ and $\bar{G}_2 = (\bar{V}_2, \bar{E}_2)$, we define a *unique* alphabet $(\Sigma', \preceq')$ obtained by jointly sorting the $(\gamma_{\min_u}, \mathtt{t}_{\min_u})$'s and the $(\gamma_{\max_u}, \mathtt{t}_{\max_u})$'s, which is possible because Lemma 17 also applies to maxima. Note that $|\bar{V}_1| + |\bar{V}_2| \leq (|V_1| + |V_2|)/2$.

Assume that we have recursively obtained the min/max-partition of $(\bar{V}_1, \bar{V}_2)$ with respect to $\bar{G}_1$ and $\bar{G}_2$. This yields the min/max-partition of $(\{u \in V_1 \mid \tau(\min_u) = 3\}, \{u \in V_2 \mid \tau(\max_u) = 3\})$. Then, we can build the min/max-partition of (V_1, V_2) by jointly applying the merging step, which is possible because both the merging step for minima and the merging step for maxima require to build the $\mathcal{A}_{c,1}$'s by processing $A_{c_1,2}A_{c_1,3}$, $A_{c_2,1}$, $A_{c_2,2}$, $A_{c_2,3}$, $A_{c_3,1}$ and so on in this order.

Since we obtain the same recursion as before, we conclude that we can compute the min/max partition of (V_1, V_2) in $O((|V_1| + |V_2|)^2)$ time.

References

1. Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Regular languages meet prefix sorting. In Shuchi Chawla, editor, *Proc. of the 31st Symposium on Discrete Algorithms, (SODA'20)*, pages 911–930. SIAM, 2020. doi:10.1137/1.9781611975994.55.
2. Ruben Becker, Manuel Cáceres, Davide Cenzato, Sung-Hwan Kim, Bojana Kodric, Francisco Olivares, and Nicola Prezza. Sorting Finite Automata via Partition Refinement. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.15.
3. Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In Ben Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics*, pages 225–235, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
4. M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Systems Research Center, 1994.
5. Alessio Conte, Nicola Cotumaccio, Travis Gagie, Giovanni Manzini, Nicola Prezza, and Marinella Sciortino. Computing matching statistics on Wheeler DFAs. In *2023 Data Compression Conference (DCC)*, pages 150–159, 2023. doi:10.1109/DCC55655.2023.00023.
6. Nicola Cotumaccio. Graphs can be succinctly indexed for pattern matching in $O(|E|^2 + |V|^{5/2})$ time. In *2022 Data Compression Conference (DCC)*, pages 272–281, 2022. doi:10.1109/DCC52660.2022.00035.
7. Nicola Cotumaccio. Prefix sorting dfas: a recursive algorithm, 2023. arXiv:2305.02526.
8. Nicola Cotumaccio, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Colexicographically ordering automata and regular languages - part i. *J. ACM*, 70(4), August 2023. doi:10.1145/3607471.
9. Nicola Cotumaccio and Nicola Prezza. On indexing and compressing finite automata. In Dániel Marx, editor, *Proc. of the 32nd Symposium on Discrete Algorithms, (SODA'21)*, pages 2585–2599. SIAM, 2021. doi:10.1137/1.9781611976465.153.
10. Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. In Tomáš Bureš, Riccardo Dondi, Johann Gamper, Giovanna Guerrini, Tomasz Jurdziński, Claus Pahl, Florian Sikora, and Prudence W.H. Wong, editors, *SOFSEM 2021: Theory and Practice of Computer Science*, pages 608–622, Cham, 2021. Springer International Publishing.

11 Massimo Equi, Veli Mäkinen, Alexandru I. Tomescu, and Roberto Grossi. On the complexity of string matching for graphs. *ACM Trans. Algorithms*, 19(3), April 2023. doi:10.1145/3588334.

12 M. Farach. Optimal suffix tree construction with large alphabets. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 137–143, 1997. doi:10.1109/SFCS.1997.646102.

13 P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *proc. 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 184–193, 2005. doi:10.1109/SFCS.2005.69.

14 P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proc. 41st Annual Symposium on Foundations of Computer Science (FOCS'00)*, pages 390–398, 2000. doi:10.1109/SFCS.2000.892127.

15 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1), November 2009. doi:10.1145/1613676.1613680.

16 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, July 2005. doi:10.1145/1082036.1082039.

17 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for BWT-based data structures. *Theoretical Computer Science*, 698:67–78, 2017. Algorithms, Strings and Theoretical Approaches in the Big Data Era (In Honor of the 60th Birthday of Professor Raffaele Giancarlo). doi:10.1016/j.tcs.2017.06.016.

18 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

19 Ramana M. Idury and Michael S. Waterman. A new algorithm for DNA sequence assembly. *Journal of computational biology: A journal of computational molecular cell biology*, 2 2:291–306, 1995.

20 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, November 2006. doi:10.1145/1217856.1217858.

21 Dong Kyue Kim, Jeong Seop Sim, Heejin Park, and Kunsoo Park. Constructing suffix arrays in linear time. *Journal of Discrete Algorithms*, 3(2):126–142, 2005. Combinatorial Pattern Matching (CPM) Special Issue. doi:10.1016/j.jda.2004.08.019.

22 Sung-Hwan Kim, Francisco Olivares, and Nicola Prezza. Faster prefix-sorting algorithms for deterministic finite automata. In Laurent Bulteau and Zsuzsanna Lipták, editors, *34th Annual Symposium on Combinatorial Pattern Matching, CPM 2023, June 26-28, 2023, Marne-la-Vallée, France*, volume 259 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CPM.2023.16.

23 Pang Ko and Srinivas Aluru. Space efficient linear time construction of suffix arrays. *Journal of Discrete Algorithms*, 3(2):143–156, 2005. Combinatorial Pattern Matching (CPM) Special Issue. doi:10.1016/j.jda.2004.08.002.

24 Veli Mäkinen, Niko Välimäki, and Jouni Sirén. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11:375–388, 2014.

25 U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.

26 Joong Chae Na. Linear-time construction of compressed suffix arrays using o(n log n)-bit working space for large alphabets. In Alberto Apostolico, Maxime Crochemore, and Kunsoo Park, editors, *Combinatorial Pattern Matching*, pages 57–67, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

27 Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987. doi:10.1137/0216062.

28 Simon J. Puglisi, W. F. Smyth, and Andrew H. Turpin. A taxonomy of suffix array construction algorithms. *ACM Comput. Surv.*, 39(2):4–es, July 2007. doi:10.1145/1242471.1242472.

29 Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory Comput. Syst.*, 41(4):589–607, 2007. doi:10.1007/s00224-006-1198-x.

30 Jared T. Simpson and Richard Durbin. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, 26(12):i367–i373, June 2010. doi:10.1093/bioinformatics/btq217.

31 P. Weiner. Linear pattern matching algorithms. In *Proc. 14th IEEE Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973. doi:10.1109/SWAT.1973.13.

Clustering in Polygonal Domains

Mark de Berg
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Leyla Biabani
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Morteza Monemizadeh
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Leonidas Theocharous
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Abstract

We study various clustering problems for a set D of n points in a polygonal domain P under the geodesic distance. We start by studying the discrete k-median problem for D in P. We develop an exact algorithm which runs in time $\text{poly}(n, m) + n^{O(\sqrt{k})}$, where m is the complexity of the domain. Subsequently, we show that our approach can also be applied to solve the k-center problem with z outliers in the same running time. Next, we turn our attention to approximation algorithms. In particular, we study the k-center problem in a simple polygon and show how to obtain a $(1+\varepsilon)$-approximation algorithm which runs in time $2^{O(k \log k/\varepsilon)}(n \log m + m)$. To obtain this, we demonstrate that a previous approach by Bădoiu *et al.* [5, 4] that works in $\mathbb{R}^d$, carries over to the setting of simple polygons. Finally, we study the 1-center problem in a simple polygon in the presence of z outliers. We show that a coreset C of size $O(z)$ exists, such that the 1-center of C is a 3-approximation of the 1-center of D, when z outliers are allowed. This result is actually more general and carries over to any metric space, which to the best of our knowledge was not known so far. By extending this approach, we show that for the 1-center problem under the Euclidean metric in $\mathbb{R}^2$, there exists an ε-coreset of size $O(z/\varepsilon)$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases clustering, geodesic distance, coreset, outliers

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.23

Funding MdB and LT are supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.

1 Introduction

Given a set D of n points in $\mathbb{R}^d$, the 1-center problem asks to find a point $p \in \mathbb{R}^d$ that minimizes the maximum distance from p to the points in D. The problem dates back to 1857, when Sylvester posed this question for the Euclidean plane [20]. A linear-time algorithm for the problem was first proposed by Megiddo [15], thus refuting an earlier conjecture of Shamos and Hoey that it cannot be solved faster than $O(n \log n)$ [19]. A natural way to generalise the 1-center problem, is to instead ask for a set $S \subset \mathbb{R}^d$ of k *centers* that minimizes the maximum distance of points in D from their closest center in S. The 1-center problem serves as a basic example of a facility location problem and is thus directly related to clustering.

In this paper, we are interested in studying this and similar clustering problems for sets of points in simple polygons and polygonal domains under the *geodesic distance*, that is, when the distance between any two points is the Euclidean length of a shortest path between them. In the literature, the term *obstructed distance* has also been used in the past to highlight that a polygonal domain can be used to model a physical environment where obstacles may obstruct or delay communication between different locations of the environment. For

example, as mentioned in [23], when a bank decides where to place ATMs, it is important to take into account the existence of highways which act as obstacles for pedestrians. Motivated by such applications, various practical clustering algorithms for realistic scenarios have been proposed [10, 21, 23, 24, 25]. Thus, the use of geodesic distance is well motivated from an application point of view.

We first study the discrete k-median and k-center problems in a polygonal domain, i.e. a polygon P with holes. The *discrete k-median problem* asks to find a set $S \subset D$ of k *center points* such that the quantity $\sum_{d \in D}\{\min_{s \in S}\{\|\pi(d,s)\|\}\}$ is minimized, where $\|\pi(p,q)\|$ denotes the length of the shortest path $\pi(p,q)$ between two points p, q in P. The *k-center problem* asks to find a set $S \subset \mathbb{R}^d$ of k center points such that the maximum distance of points in D to their nearest center in S is minimized (so here, in contrast with k-median, we consider the continuous version of the problem). An outlier can significantly increase the maximum distance to the nearest center, so we also study the *k-center problem with z outliers*, which asks to minimize the maximum distance of all but z points of D to their nearest center. Our interest here lies in developing an *exact* algorithm for these problems, whose running time is polynomial in n and m (for fixed k), and whose dependency on k is subexponential. (Note that the Euclidean problem in the plane is already NP-hard when k is part of the input [16], so an algorithm that is also polynomial in k is not possible, assuming P $\neq$ NP.) The k-center problem in the plane has been studied extensively, for general k [9, 13] and also for the special case $k = 2$ [1, 22]. Most relevant to our approach is the work by Hwang *et al.* [12], who presented algorithms with running time $n^{O(\sqrt{k})}$ for the Euclidean version of the problems in $\mathbb{R}^2$. Their approach works with the Voronoi diagram of the (unknown) optimal solution, and "guesses" a cycle separator of its dual graph. The separator splits the problem into two subproblems, which are then solved recursively. This is an idea that we also make use of. The same approach was employed more recently by Marx and Pilipczuk [14] to solve a wide range of covering and packing problems defined on planar graphs. This includes k-center, which they solve in $n^{O(\sqrt{k})}$. To the best of our understanding, their approach cannot be used to tackle k-median and also cannot directly handle outliers. Thus, in Section 2 we develop an exact algorithm for the discrete k-median problem in a polygonal domain. The running time is $\text{poly}(n,m) + n^{O(\sqrt{k})}$, where m is the complexity of the domain. With our approach, we can also solve the k-center problem with z outliers in the same running time.

Next, we develop an FPT approximation algorithm for the k-center problem in a simple polygon, that is, an algorithm whose running time is $O(f(k,\varepsilon) \cdot \text{poly}(n,m))$, for some computable function f. Towards this algorithm, we first study the 1-center problem. Exact algorithms for the 1-center problem in a simple polygon have been developed before. More specifically, Ahn *et al.* [2] studied the problem of computing the *geodesic center* of a (weakly) simple polygon P, where the task is to find the point $s \in P$ that minimizes the maximum geodesic distance from *any* other point in P. They developed a linear-time algorithm for this problem. Their algorithm can be used to compute the 1-center of a set D of n points in P because the 1-center of D coincides with the geodesic center of $\text{RCH}_P(D)$, the relative convex hull of D in P. Since the geodesic convex hull is a weakly simple polygon that can be computed in time $O(n \log n + m)$, where m is the complexity of P), the 1-center of D can be computed in the same time.

Here, however, we study this problem through the lens of coresets. In general, a coreset is a small subset of the input to a problem, such that the solution of the problem on the coreset is a good approximation of the solution on the whole input. Coresets can be categorised in *strong coresets* and *weak coresets*, depending on the kind of guarantee they provide. Roughly

speaking, a strong coreset provides error guarantees for *any* candidate solution on the coreset, while a weak coreset only guarantees that an optimal solution on the coreset is a good approximation of an optimal solution on the whole set. All coresets presented in this paper are weak coresets, so from now on we will just use the term coreset.

Most related to our work are previous approaches for constructing coresets for the 1-center problem in $\mathbb{R}^d$. Bădoiu *et al.* [5] showed the existence of a coreset $C \subset D$ of size $O(\frac{1}{\varepsilon^2})$, such that the 1-center of the coreset is a $(1+\varepsilon)$-approximation to the 1-center of a set of points $D \subset \mathbb{R}^d$. The time to construct this coreset is $O\left(\frac{dn}{\varepsilon^2} + \frac{1}{\varepsilon^{O(1)}}\right)$. They then showed that their approach can be extended to obtain a $(1+\varepsilon)$-approximation for the k-center problem, for $k > 1$ in time $2^{O(k \log k/\varepsilon^2)}dn$. By providing a better analysis of the approach in [5], Bădoiu and Clarkson [4] showed that the coreset obtained actually has size $O(1/\varepsilon)$, which is tight. We show that these approaches also work when the underlying space is a simple polygon. A priori this is not at all clear, because the geodesic metric in a simple polygon does not have bounded doubling dimension. Specifically, in Section 3 we show the existence of an ε-coreset of size $O(1/\varepsilon)$ for the 1-center problem for a set of points in a simple polygon. The time to construct this is $O\left(\frac{n \log m + m}{\varepsilon} + \frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon}\right)$, where m is the complexity of the polygon. Note that a coreset of size two or three always exists in the plane (both in the Euclidean setting as well as in a polygon), since the minimum enclosing (geodesic) ball of a set of points in P is defined by two or three of the points. Thus it can be computed in the same time it takes to compute the 1-center of D, which, as mentioned, is $O(n \log n + m)$. Hence, for constant m the construction takes $O(n \log n)$ time, whereas our coreset can be constructed in $O\left(n/\varepsilon + 1/\varepsilon^2 \log(1/\varepsilon)\right)$ time. More importantly, our coreset can be combined with the approach by Bădoiu et al. [5] to approximately solve the k-center problem for $k > 1$, in $2^{O(k \log k/\varepsilon)}(n \log m + m)$ time.

Finally, we study the 1-center problem with z outliers through the lens of coresets in Section 4. We show that in any metric space, there exists a coreset of size $2z+2$ that is a 3-approximation for the 1-center problem with z outliers. In the Euclidean plane, we can generalize our result and obtain an ε-coreset of size $O(z/\varepsilon)$. Coresets for the k-center problem with z outliers have been studied for the metrics of bounded doubling dimension [7, 6]. Particularly, De Berg *et al.* [7] present an ε-coreset of size $O(k/\varepsilon^d + z)$, where d is the doubling dimension. In the plane, their construction can give an ε-coreset of size $O(k/\varepsilon^2 + z)$. Note that the dependency on $1/\varepsilon$ in their bound is quadratic, while our coreset only has a linear dependency on $1/\varepsilon$. (On the downside, in our case this is multiplied by z, while [7] has an additive term in z.) They also show that under some natural conditions, any coreset with a constant approximation ratio is of size $\Omega(k+z)$.

2 k-Median and k-center with outliers in a polygonal domain

In this section, we study the discrete k-median problem in a polygonal domain P. We will develop a subexponential exact algorithm for this problem, which depends exponentially on k and not on the complexity of the polygonal domain. We will then show that our algorithm can also be used to solve the k-center problem with outliers in a polygonal domain. We start by introducing some notation.

Notation. We denote the outer polygon of our polygonal domain P by P_0, and we use $\mathcal{H}$ to denote the collection of holes in P. Recall that $\pi(p,q)$ denotes the shortest path between two points p, q in P. For a finite set $D \subset P$, the *geodesic Voronoi diagram* of D in P, denoted $\text{GVD}(D)$, is the partition of P into $|D|$ Voronoi cells, where the Voronoi cell $V(q)$ of a point

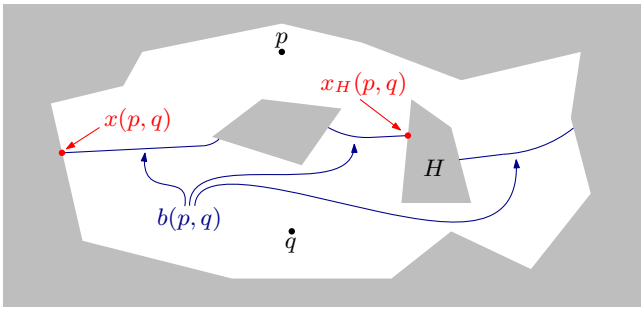


Figure 1 Illustration for the definition of $b(p,q), x(p,q)$ and $x_H(p,q)$.

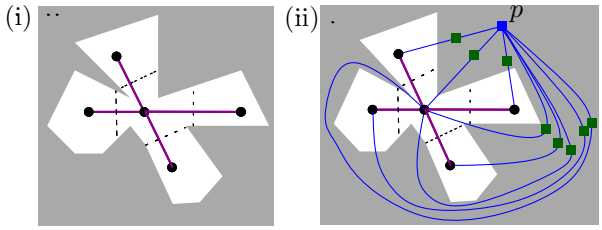


Figure 2 (i) An example where the dual of the geodesic Voronoi diagram corresponds to a tree. (ii) Adding p to the outside face of P and connecting it to cells incident to ∂P_0 via the intervals I_i.

$q \in D$ is defined as $V_D(q) := \{x \in P : \|\pi(x,q)\| \leqslant \|\pi(x,p)\| \text{ for all } p \in D\}$. When the set D is clear from the context, we may simply write $V(q)$. For two points $p, q \in P$, let $b(p,q)$ denote their geodesic bisector and let $b_D(p,q)$ denote the part of $b(p,q)$ which appears in $\text{GVD}(D)$. Let $B(p,q) = \{x \in P : \|\pi(p,x)\| \leqslant \|\pi(q,x)\|\}$. We will denote by $x(p,q)$ the first point of $b(p,q)$ that is met during a clockwise transversal of ∂P_0 which starts from a point of $\partial P_0 \cap B(p,q)$. Finally, we will denote by $x_H(p,q)$ the first point of $b(p,q)$ that is met during a clockwise transversal of hole H, starting from a point of $H \cap B(p,q)$; see Figure 1.

The main idea. The idea is to extend the approach by Hwang *et al.* [12], which worked for $\mathbb{R}^2$, to a polygonal domain. We therefore start by considering the geodesic Voronoi diagram of our (unknown) optimal solution S. In the Euclidean case, the dual of this diagram is called the Delaunay triangulation, denoted by $\text{DT}(S)$. Every inner face of $\text{DT}(S)$ is a triangle, and one can add a set I of three extra points to S sufficiently far away, such that the outside face of $\text{DT}(S \cup I)$ also becomes a triangle. This results in a maximal planar graph. Hence, by Miller's separator theorem [17] there exists a simple cycle separator C of $\text{DT}(S \cup I)$ of size $O(\sqrt{k})$ which is (2/3)-balanced with respect to $S \cup I$. (The latter means that at most 2/3 of the points in $S \cup I$ lie inside C and at most 2/3 of the points in $S \cup I$ lie outside C.) In our setting, it is not guaranteed that the dual of $\text{GVD}(S)$ is an (almost) triangulated graph. See Figure 2(i) for an example where it corresponds to a tree. Therefore we will need a few extra steps before we can apply a separator theorem.

Transforming the dual of $\text{GVD}(S)$. Let $\mathcal{G} = (V, E)$ denote the dual graph of $\text{GVD}(S)$. The goal is to transform $\mathcal{G}$ to a graph $\mathcal{G}^* = (V^*, E^*)$ such that any face of $\mathcal{G}^*$ has size at most three. The Voronoi cells of $\text{GVD}(S)$ that are incident to ∂P_0, induce a decomposition of ∂P_0 into disjoint intervals. Note that it is possible for a Voronoi cell to contribute to more than one interval. The following lemma gives a linear bound on the number of these intervals.

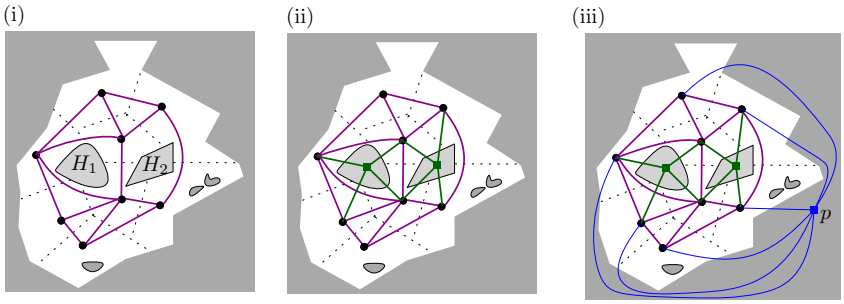


Figure 3 Holes H_1 and H_2 are essential, so we add a vertex for each of them. In (iii), observe that every face has bounded size.

▶ **Lemma 2.1.** *Let $I_1, \ldots, I_r$ denote the intervals along ∂P_0 induced by* GVD(S)*, enumerated in clockwise order. Then $r = O(k)$.*

Proof. For $1 \leqslant i \leqslant r$, let $s_i \in S$ be the center in the optimal solution S whose Voronoi cell has I_i on its boundary. Note that the s_i need not all be distinct. For $i = 1, ...r-1$, we charge I_i to $b(s_i, s_{i+1})$. Any bisector can be charged at most two times. Moreover, a bisector uniquely corresponds to an edge of $\mathcal{G}$ and we know that $\mathcal{G}$ is a planar graph. Therefore $r \leqslant 2|E| \leqslant 6k - 12$. ◀

Now let p denote an arbitrary point in the outside face of P. We connect p to each s_i via any arbitrary interior point of I_i. Let $\{e_1, \ldots, e_r\}$ denote the set of these extra edges. Then we have so far, $V^* = V \cup \{p\}$ and $E^* = E \cup \{e_i\}_{i=1}^r$. It's easy to see that we can embed these edges such that: (i) they are pairwise non-crossing and (ii) any face of the resulting graph incident to p is a triangle. See Figure 2(ii) for an example.

Handling the faces that do not contain p. Now we need to handle the faces of $\mathcal{G}^*$ that are not incident to p. By construction, the outer face of $\mathcal{G}^*$ contains p and thus is a triangle. Therefore, the only way $\mathcal{G}^*$ can contain a face of size at least four is if there exists a cycle of size four "around" a hole as in Figure 3(i).

We define an *essential hole* to be a hole $H \in \mathcal{H}$ which is incident to at least four Voronoi cells of GVD(S). Since every essential hole corresponds to a face of $\mathcal{G}$ (or $\mathcal{G}^*$), the number of essential holes is $O(k)$. Let $\mathcal{H}^*$ denote the set of essential holes and for every $H \in \mathcal{H}^*$ let p_H be an arbitrary point in H. We add the set $\{p_H\}_{H \in \mathcal{H}^*}$ to V^* and we connect p_H to the vertices of the Voronoi cells that are incident to H. If $V(q)$ is such a Voronoi cell, then, as before, we can embed the edge (p_H, q) by going through any interior point of $H \cap V(q)$. At the end of this process, $\mathcal{G}^*$ is a graph where every face is a triangle. See Figure 3(iii).

2.1 Applying the Separator Theorem to $\mathcal{G}^*$

We now want to apply Miller's Separator Theorem to $\mathcal{G}^*$. One thing that prevents us from doing so, is that $\mathcal{G}^*$ could be a multigraph, because p may be connected to the same Voronoi vertex more than once. (Recall that a Voronoi cell may contribute to more than one interval I_i). To deal with this, we can add a "dummy vertex" to each edge which has p as an endpoint; see Figure 2(ii). This way we only increase the number of vertices and edges by $O(k)$. Moreover, the faces of the resulting graph still have bounded size, which ensures that a separator theorem can still be applied (see below). Note that we want our separator to be balanced with respect to V. To ensure that, we employ the cost-balanced version of the Planar Separator Theorem, proven by Djidjev and Venkatesan [8].

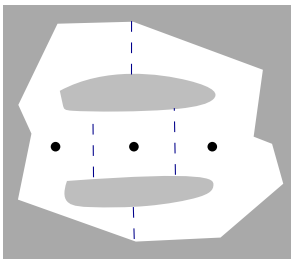


Figure 4 An example of three points and two holes, such that all three pairwise bisectors between the points intersect both holes.

> **Planar Separator Theorem.** Let $G = (\mathcal{V}, \mathcal{E})$ be a maximal planar graph with n nodes. Let each node $v \in \mathcal{V}$ have a non-negative weight, denoted weight(v), with $\sum_{v \in V}$ weight$(v) = 1$. Then $\mathcal{V}$ can be partitioned in $O(n)$ time into three sets A, B, C such that (i) C is a simple cycle of size $O(\sqrt{n})$, (ii) G has no arcs between a node in A and a node in B, and (iii) $\sum_{v \in A}$ weight$(v) \leqslant 2/3$ and $\sum_{v \in B}$ weight$(v) \leqslant 2/3$.

The theorem is stated for maximal planar graphs, but as pointed out in [8], it can be extended to graphs with faces of bounded size (as is our case). In our application, we give weight zero to the intermediate vertices as well as all vertices in $V^* \setminus V$, and weight $\frac{1}{|V|}$ to each vertex in V. Thus we obtain a simple-cycle separator C, which we can turn into a separator for $\mathcal{G}^*$ by ignoring any of the dummy vertices appearing on it. We obtain the following lemma.

▶ **Lemma 2.2.** *There exists a separator C for $\mathcal{G}^*$ with the following properties: **1.** C is a simple cycle, **2.** C has size $O(\sqrt{k})$ and **3.** C is $(2/3)$-balanced with respect to V.*

2.2 Guessing and embedding the separator

What we would like to do now, is guess the separator C of $\mathcal{G}^*$. Regarding the essential holes, note that we know that $|\mathcal{H}^*| = O(k)$, but we don't have any bound on $\mathcal{H}$ in terms of n, the size of the input point set D. This is problematic for the running time because we will need to "guess" what the essential holes are. However, we will argue that only $O(n^3)$ holes are good candidates for being essential. We start with the following lemma.

▶ **Lemma 2.3.** *Let P be a polygon and let $\mathcal{H} = \{H_1, H_2, ..., H_m\}$ be the set of holes in P. Let $T = \{p, q, r\}$ be a set of three points in P. Then there are at most two holes in $\mathcal{H}$ that are incident to all three Voronoi cells $V_T(p), V_T(q), V_T(r)$.*

Proof. Assume for contradiction that there exist three holes H_i, H_j, H_k incident to all three cells $V_T(p), V_T(q), V_T(r)$. Since Voronoi cells are connected, for each $x \in \{p, q, r\}$ and for each $H \in \{H_i, H_{,j}, H_k\}$, there exists a path connecting x to H which stays inside $V(x)$. In this way, we get a planar embedding of $K_{3,3}$, which is a contradiction. (Note that it *is* possible to have two holes bordering $V_T(p), V_T(q), V_T(r)$, see Figure 4.) ◀

Now we define the set of *candidate essential holes* $\mathcal{R}$ as follows: for every triplet of points in D we identify at most two holes which are incident to all three pairwise bisectors between the points. We then place these holes in $\mathcal{R}$. Clearly, $|\mathcal{R}| = O(n^3)$. Therefore we can afford to guess $O(\sqrt{k})$ essential holes on our separator C.

Now we give a more detailed description of how our algorithm works. Recall that each node in $\mathcal{G}^*$ (and, hence, each node on the separator we are looking for) corresponds to either a point in D, or to the extra point p we added, or to an essential hole. Thus, to find the

separator, we guess all ordered subsets of size $O(\sqrt{k})$ from the set $R \cup D \cup p$. This results in $n^{O(\sqrt{k})}$ candidate separators. We then would like to use each separator to split our problem into two independent subproblems, one for the inside and one for the outside of the separator. To do that, we have to make sure that for every demand point $d \in D$ its closest center point in the optimal solution, is located at the same side of the separator as d. For this, it suffices to embed the edges of the separator such that no edge crosses a Voronoi cell of a point which is not one of its endpoints. We have three categories of edges:

- **Edges that connect p to a Voronoi site s.** Clearly if $(p, s) \in E^*$, then there exists some $r \in S$ such that $(s, r) \in E$ and then we can embed such an edge using the shortest paths $\pi(p, x(s, r)), \pi(x(s, r), s)$. Note that we don't know r, but we can afford to guess it from D and therefore there are n options.
- **Edges that connect a Voronoi site s to a p_H for some $H \in \mathcal{R}$.** If $(s, p_H) \in E^*$, then again there exists some $r \in S$ such that $s, r \in E$ and then we can embed such an edge by going through $x_H(s, r)$, again using shortest paths. We can guess r from D and there are n options.
- **Edges that connect two Voronoi sites s and r.** Note that then one of the following holds if $(s, r) \in E$:
  1. there exists a $t \in S$, such that $V_S(s), V_S(r), V_S(t)$ meet at a point c in P or at a hole $H \in H^*$
  2. $b_S(s, r)$ intersects ∂P_0 at two points.

  Therefore we can check for all $t \in D$ whether 1. holds and if yes we embed (s, r) as $\pi(s, c) \cup \pi(c, r)$ or as $\pi(s, x_H(s, r)) \cup \pi(x_H(s, r), r)$. Otherwise, we can embed (s, r) via $x(s, r)$. Again we have at most n guesses.

Since we are using shortest paths to embed the edges, we know that they will stay inside the Voronoi cells of the sites they connect and thus we get a good embedding. Assuming our guessed separator is correct, the points on the separator have to be part of the optimal solution that we seek. Therefore in the two subproblems, these points have to be passed on as part of the input. If we assume that our separator has size i then we also need to guess how many of the remaining $k - i$ optimal centers lie in the inside and how many lie in the outside subproblem. In terms of running time, this is clearly not a problem since it can only give an extra factor of $O(k) = O(n)$. The base case of our algorithm is when $k = 1$, where we simply try all possible options.

A word on precomputing shortest paths. Our algorithm will need to make use of shortest paths between points in the set $D \cup \{p\} \cup \{p_H\}_{H \in \mathcal{R}} \cup \{x(p, q), x_H(p, q)\}_{(p,q,H) \in D \times D \times \mathcal{R}}$. Note that this set has size $\text{poly}(n)$ and thus all necessary shortest paths can be precomputed in $\text{poly}(n, m)$ time. The only information about these paths our algorithm will need during the recursion is their length and whether two paths cross or not. Indeed, this is enough to determine, for a guessed separator C, which are its two corresponding subproblems; two points p, q not on C will be on different sides of C if and only if $\pi(p, q)$ crosses C an odd number of times. Therefore, the complexity of the polygon does not appear during the recursion. Note that the same idea was used for the preprocessing step in [3].

Given the above discussion, the recursive formula of our algorithm is of the form $T(k) = n^{O(\sqrt{k})} T(2k/3)$, which solves to $T(k) = n^{O(\sqrt{k})}$. The following theorem summarises our result, where the $\text{poly}(n, m)$ term comes from precomputing shortest paths.

▶ **Theorem 2.4.** *Let D be a set of n points inside a polygonal domain P with m vertices and let k be a given positive integer. Then the discrete k-median problem for D in P can be solved in time* $\text{poly}(n, m) + n^{O(\sqrt{k})}$.

2.3 The k-Center problem with outliers

To solve the k-center problem with z outliers, we first show that the same approach as our k-median algorithm works to solve the so-called (k, r)-coverage problem, and then we show how to reduce the k-center problem with z outliers to the (k, r)-coverage problem. Let P be a polygonal domain, D denote a set of n demand points in P, and k, r be two parameters. We define a (k, r)-*coverage of* D as a set of k balls of radius at most r, such that the number of outliers (that is, points in D not covered by the balls) is minimized.

The divide-and-conquer algorithm we presented earlier for k-median clustering has a base case of $k = 1$. Our algorithm relies on the n candidates for the optimal centers in k-median, and has a running time of $n^{O(\sqrt{k})}$. However, we only use the properties of k-median to solve for the base case and determine the candidate centers when guessing the separator in an optimal solution. Note that above we solved the *discrete* k-median problem, where the set of candidate centers is given (namely, it is the same as the set D of demand points, although our algorithm would also work if a different discrete set of candidate centers is given). For the k-center problem, we wish to solve the *continuous* version, where the set of candidate centers is not given. It is well known, however, that in the continuous k-center problem, we can still restrict our attention to a discrete set of candidates, namely the centers of the smallest enclosing (geodesic) balls of every triple and pair of points in D. Thus there are $O(n^3)$ candidate centers. We denote the set of candidate centers by C^*. Therefore, the same approach can be applied to solve the (k, r)-coverage problem, where for the base case, we can consider all $O(n^3)$ balls of radius r centered at a point in C^* and find the one that covers the maximum number of points in D. This means that our algorithm can compute an optimal (k, r)-coverage in $n^{O(\sqrt{k})}$ time.

It remains to reduce the k-center problem with z outliers to the (k, r)-coverage problem. Observe that if r is at least the optimal radius for k-center clustering with z outliers, then the number of outliers for (k, r)-coverage is at most z. Moreover, there are at most $O(n^3)$ candidates for the optimal radius (namely the radii of the smallest enclosing balls of the triples and pairs of points in D). By performing a binary search over these $O(n^3)$ possible radii, we can find the minimum radius r^* such that the (k, r^*)-coverage covers all but at most z outliers. This (k, r^*)-coverage is an optimal solution for the k-center problem with z outliers, and the running time to find it is $O(n^{O(\sqrt{k})} \cdot \log(n^3)) = n^{O(\sqrt{k})}$.

▶ **Theorem 2.5.** *Let D be a set of n points inside a polygonal domain P with m vertices and let k, z be two given integers. Then the k-center problem for D with z outliers can be solved in time* $\text{poly}(n, m) + n^{O(\sqrt{k})}$.

3 A coreset for the k-center of points in a simple polygon

In this section, we turn our attention to the k-center problem in a simple polygon. As already mentioned, here we are interested in coresets for this problem. We will start by studying the 1-center. In itself, a coreset for the 1-center in a simple polygon is not so interesting, since the minimum enclosing (geodesic) ball of a set D of points inside P is always defined by two or three points, and so there exists a coreset of size two or three. However, the technique that we use (which is borrowed from Bădoiu and Clarkson [4]) forms the basis of the result for k-center. For a set S of points, let c_S denote the center of the minimum enclosing geodesic ball of S (that is, its 1-center) and let r_S be its radius. We denote the ball of radius r centered at a point c be $B(c, r)$, so $B(c_S, r_S)$ is the minimum enclosing ball of S. Note that these definitions apply in the standard Euclidean case, but also for geodesic

distances in a simple polygon. Our algorithm to compute a coreset $\mathcal{C}$ for the 1-center of a point set D inside a simple polygon P uses the approach of Bădoiu *et al.* [5, 4], which works as follows. First, we place in $\mathcal{C}$ an arbitrary point $p \in D$. Then we repeat the following procedure: we check whether there exists a point $q \in D$ such that $\|\overline{c_{\mathcal{C}}q}\| > (1+\varepsilon)r_{\mathcal{C}}$ and if yes, we add in $\mathcal{C}$ the point of D furthest from $c_{\mathcal{C}}$. Otherwise, we have our desired coreset. The analysis of the number of iterations of the above procedure relies on two key lemmas. Note that any chord in a simple polygon P (that is, any segment inside P connecting two points on ∂P) splits P into two sub-polygons, which we call *half-polygons*.

▶ **Lemma 3.1.** *Let $B(c_D, r_D)$ denote the minimum enclosing geodesic ball for a set D of points inside a simple polygon P. Then any closed half-polygon containing c_D also contains a point $p \in D$ such that $\|\pi(p, c_D)\| = r_D$.*

Proof. We proceed in the same way as in the Euclidean case. Namely, suppose there exists a chord s of P through c_D, such that one of the two defined half-polygons does not contain a point $p \in D$ such that $\|\pi(p, c_D)\| = r_D$. Let H_1 denote this half-polygon. Then we can slightly move c_D to the direction perpendicular to s and to the interior of $P \setminus H_1$. In this way, every point of D will now be fully contained in the interior of $B(c_D, r_D)$. The reason is that any shortest path $\pi(c_D, q)$, for $q \in P \setminus H_1$ is contained in $P \setminus H_1$ and thus this translation of c_D can only decrease $\|\pi(c_D, q)\|$. As a result the ball $B(c_D, r_D)$ can be slightly shrunk, contradicting its minimality. ◀

The second lemma we need is as follows.

▶ **Lemma 3.2.** *Let $B(c_D, r_D)$ denote the minimum enclosing geodesic ball for a set D of points inside a simple polygon P. For any point $q \in P$, there exists a point $p \in D$ at distance r_D from c_D such that $\|\pi(p, q)\| \geqslant \sqrt{\|\pi(p, c_D)\|^2 + \|\pi(c_D, q)\|^2}$.*

The corresponding lemma in the Euclidean case (that is when $P = \mathbb{R}^2$) follows directly from (the Euclidean version of) Lemma 3.1, by using the Pythagorean Inequality. For geodesic triangles however, we were not able to find in the literature an analog of the Pythagorean Inequality. Thus we prove now that the following property still holds for geodesic triangles in a simple polygon. Note that Lemma 3.3 together with Lemma 3.1 imply Lemma 3.2. Indeed, let s be the chord through c_D that is perpendicular to the first edge of $\pi(c_D, p)$. Then by applying Lemma 3.1 to the closed half-polygon defined by s and not containing q, we get that there exists a point $p \in D$ such that $\|\pi(p, c_D)\| = r_D$. The result follows by observing that in the geodesic triangle $\triangle_\pi p c_D q$, the angle at c_D is at least $\frac{\pi}{2}$ and thus Lemma 3.3 applies.

▶ **Lemma 3.3.** *Let p, q, r denote three points in a simple polygon P, such that in the geodesic triangle $\triangle_\pi pqr$, the angle at q is at least $\frac{\pi}{2}$. Then we have $\|\pi(p, r)\|^2 \geqslant \|\pi(p, q)\|^2 + \|\pi(q, r)\|^2$.*

Proof. For the following, refer to Figure 5. Let $\overline{qq_1}, \overline{qq_2}$ denote the first edges of the paths $\pi(q, p), \pi(q, r)$ respectively. We extend $\overline{qq_1}, \overline{qq_2}$ to the interior of $\triangle_\pi pqr$ and let q_1', q_2' denote the points where these extensions intersect $\pi(p, r)$ respectively. By the triangle inequality we have

$$\|\pi(p, q)\| \leqslant |qq_1'| + \|\pi(q_1', p)\| \text{ and } \|\pi(r, q)\| \leqslant |qq_2'| + \|\pi(q_2', r)\|.$$

Moreover, since the angle at q is at least $\pi/2$ and the Euclidean triangle $\Delta(qq_1'q_2')$ satisfies the Pythagorean Inequality, and $\|\pi(q_1', q_2') \geqslant |q_1'q_2'|$, we have

$$\|\pi(q_1', q_2')\|^2 \geqslant |qq_1'|^2 + |qq_2'|^2.$$

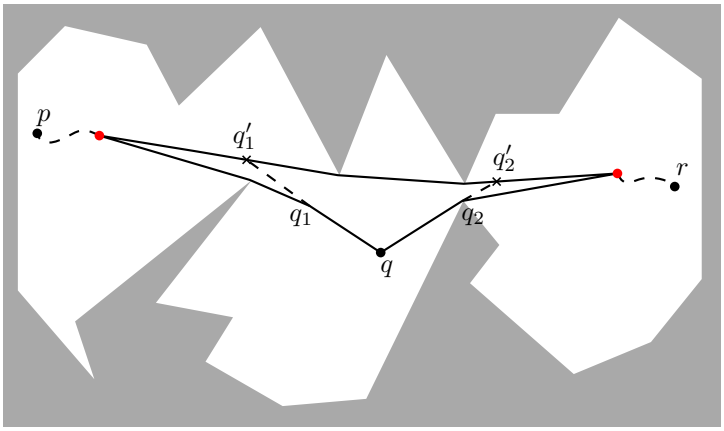


Figure 5 Illustration for the proof of Lemma 3.3. Here, the red points represent the points where the paths $\pi(p,r), \pi(p,q)$ and $\pi(r,q), \pi(r,p)$ split.

Finally, observe that for any numbers $a, b, c \geqslant 0$ we have

$$(a+b+c)^2 \geqslant a^2 + b^2 + c^2 + 2ab + 2ac.$$

Hence,

$$\begin{aligned}
\|\pi(p,q)\|^2 + \|\pi(r,q)\|^2 &\leqslant \left(|qq_1'| + \|\pi(q_1',p)\|\right)^2 + \left(|qq_2'| + \|\pi(q_2',r)\right)^2 \\
&= |qq_1'|^2 + 2|qq_1'| \cdot \|\pi(q_1',p)\| + \|\pi(q_1',p)\|^2 + |qq_2'|^2 + 2|qq_2'| \cdot \|\pi(q_2',r)\| + \|\pi(q_2',r)\|^2 \\
&= \|\pi(q_1',q_2')\|^2 + \|\pi(q_1',p)\|^2 + \|\pi(q_2',r)\|^2 + 2|qq_1'| \cdot \|\pi(q_1',p)\| + 2|qq_2'| \cdot \|\pi(q_2',r)\| \\
&\leqslant \left(\|\pi(q_1',q_2')\| + \|\pi(q_1',p)\| + \|\pi(q_2',r)\|\right)^2 \\
&= \|\pi(p,r)\|^2.
\end{aligned}$$

◀

We can now prove the following theorem.

▶ **Theorem 3.4.** *Let D be a set of n points inside a simple polygon P with m vertices. For any $\varepsilon > 0$ there is an ε-coreset $\mathcal{C} \subset D$ of size $O(1/\varepsilon)$ for the 1-center problem. The coreset can be constructed in time $O\left(\frac{n \log m + m}{\varepsilon} + \frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon}\right)$.*

Proof. The proof is similar to that of Badoiu *et al.* [4]: let $\mathcal{C}_i$ denote our coreset after i points have been added to it and let $B(r_i, c_i)$ denote its minimum enclosing ball. Observe that r_2 is a constant approximation for the optimal radius. Therefore, if one can show that $r_{i+1} \geqslant \left(1 + \frac{\varepsilon}{\alpha}\right) r_i$, for some constant α, it will follow that after $O(1/\varepsilon)$ iterations, $r_{\mathcal{C}} \geqslant r_D$. By applying Lemma 3.2 we get $r_{i+1} \geqslant \sqrt{r_i^2 + \|\pi(c_{i+1}, c_i)\|^2}$. By the triangle inequality we get $r_{i+1} > (1+\varepsilon) r_i - \|\pi(c_{i+1}, c_i)\|$. Using these two lower bounds for r_{i+1}, one can indeed get the desired relation between r_{i+1} and r_i. For further details, refer to the proof in [4].

Regarding the construction of the coreset, if we follow the procedure described, then we need to solve the 1-center problem $O\left(\frac{1}{\varepsilon}\right)$ times for a set of $O(\frac{1}{\varepsilon})$ points in a simple polygon with m vertices. This can be done in $O\left(\frac{1}{\varepsilon}\left(m + \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)\right)$ time [18]. In each iteration, we also need to find the point in D that is furthest from our current center. This takes $O(n \log m)$ time per iteration [11]. Hence, the total running time is $O\left(\frac{n \log m + m}{\varepsilon} + \frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon}\right)$. ◀

Extension to k-center. Theorem 3.4 combined with the methods from [5] yields a $(1+\varepsilon)$-approximate k-center for D. The algorithm starts with k, initially empty, sets $S_1, S_2, ..., S_k$. In each iteration, the point $p \in D$ furthest from $c_{S_1}, c_{S_2}, ..., c_{S_k}$ is found and added to

one of the sets. However, we do not know beforehand to which set the point p should be added, so we simply guess (that is, try all possibilities). After $O(k/\varepsilon)$ iterations, the algorithm will terminate in the branch where all guesses of where the furthest point should be added are correct. Therefore we also need to guess to which set to add p. We obtain the following theorem, whose proof is the same as the proof of the corresponding result of Bădoiu, Har-Peled, and Indyk [5].

▶ **Theorem 3.5.** *Let D be a set of n points inside a simple polygon P with m vertices. For any $1 > \varepsilon > 0$, a $(1+\varepsilon)$- approximate k-center for D can be found in time $2^{O(k \log k/\varepsilon)}(n \log m + m)$.*

4 Coresets for 1-center clustering with outliers

We now study the 1-center problem with outliers for points in the Euclidean plane (so, not inside a polygon). We will prove that there is an ε-coreset of size $O(z/\varepsilon)$ for this problem. To obtain such a coreset, our algorithm will need a constant-factor approximation as a starting point. To this end, we first show how to construct a coreset of size $2z + 2$ that gives a 3-approximation. Interestingly, this latter algorithm works in any metric space, so also for points inside a polygonal domain.

A coreset giving a 3-approximation in general metric spaces. Let D be a set of points in a metric space with distance function $d(\cdot, \cdot)$. We show in Algorithm 1 how to compute a coreset of size $2z + 2$ that 3-approximates the 1-center problem on D with z outliers.

Note that if the size of D is at most $2z + 2$, we return D in Algorithm 1 as the coreset. Therefore, without loss of generality, we can assume that the size of D is greater than $2z + 2$ for the rest of this section. We define $\text{OPT}_z(A)$ as the radius of an optimal solution for the 1-center problem on D with z outliers, where A represents any given set.

Algorithm 1 FindCoreset(D, z).

```
1: ▷ An algorithm to find a coreset for 1-center problem on D with z outliers
2: if |D| ⩽ 2z + 2 then
3:     return D
4: C_1 ← {an arbitrary point of D}
5: for i = 2 to 2z + 1 do
6:     Let B(o_{i−1}, r_{i−1}) be a minimum-radius ball containing all points of C_{i−1} but z outliers
7:     Let f_i be a point of D \ C_{i−1} that is furthest away from o_{i−1}
8:     C_i ← C_{i−1} ∪ {f_i}
9: return C_{2z+2}
```

▶ **Lemma 4.1.** *For any $1 \leqslant i \leqslant 2z + 2$, at least one of the following properties holds for the set C_i constructed by Algorithm FindCoreset(D, z).*

(i) $\text{OPT}_z(C_i) \geqslant \text{OPT}_z(D)/3$, *or*

(ii) *for all $r < \text{OPT}_z(D)/3$, any ball of radius r contains at most $z + 1$ points of C_i.*

Proof. For $i = 1$, property (ii) is trivially satisfied since we have $|C_i| = 1$ and $z + 1 \geqslant 1$. For $i > 1$, we prove the lemma by contradiction. Suppose the lemma is false, and let $t \in [2, 2z+2]$ be the smallest number such that properties (i) and(ii) do not hold for C_t. Then, there exists a ball $B(o', r')$ such that $|B(o', r') \cap C_t| > z + 1$ and $r' < \text{OPT}_z(D)/3$.

Let f_t be the point added to the coreset in line 8 of the algorithm, in the t-th iteration. We first prove that $f_t \in B(o', r')$ and $|B(o', r') \cap C_{t-1}| = z + 1$. Since $C_t \supset C_{t-1}$, we have $\text{OPT}_z(C_{t-1}) \leqslant \text{OPT}_z(C_t)$. Moreover, we assumed that property (i) does not hold for C_t,

which means $\text{OPT}_z(C_t) < \text{OPT}_z(D)/3$. Therefore, $r_{t-1} = \text{OPT}_z(C_{t-1}) < \text{OPT}_z(D)/3$, which means property (i) does not hold for C_{t-1}. As t is the smallest number such that both properties do not hold for C_t, we conclude that property (ii) holds for C_{t-1}, which implies $|B(o', r') \cap C_{t-1}| \leqslant z+1$. Adding it to $|B(o', r') \cap C_t| > z+1$ and $C_t = C_{t-1} \cup \{f_t\}$ we have $f_t \in B(o', r')$ and also $|B(o', r') \cap C_{t-1}| = z+1$.

$B(o_{t-1}, r_{t-1})$ is a solution for the 1-center problem on C_{t-1} with z outliers, and $|B(o', r') \cap C_{t-1}| = z+1$. Then, there exists a point $p^* \in B(o_{t-1}, r_{t-1}) \cap (C_{t-1} \cap B(o', r'))$. Moreover, by the triangle inequality we have $d(o_{t-1}, f_t) \leqslant d(o_{t-1}, p^*) + d(p^*, f_t)$. As $p^* \in B(o_{t-1}, r_{t-1})$, we have $d(o_{t-1}, p^*) \leqslant r_{t-1}$, and as both p^* and f_t are in $B(o', r')$ we have $d(p^*, f_t) \leqslant 2r'$. Thus, $d(o_{t-1}, f_t) \leqslant r_{t-1} + 2r'$, and since $r_{t-1} < \text{OPT}_z(D)/3$ and $r' < \text{OPT}_z(D)/3$ we have $d(o_{t-1}, f_t) < \text{OPT}_z(D)$.

On the other hand, we have $d(o_{t-1}, f_t) \geqslant d(o_{t-1}, p)$ for any $p \in D \setminus C_{t-1}$ since f_t is the furthest point in D/C_{t-1} to o_{t-1}. Furthermore, all but at most z points of C_{t-1} are in $B(o_{t-1}, r_{t-1})$. Also, $d(o_{t-1}, f_t) \geqslant r_{t-1}$, since otherwise $r_{t-1} = \text{OPT}_z(C_{t-1}) \geqslant \text{OPT}_z(D)$, which is a contradiction to $\text{OPT}_z(C_{t-1}) < \text{OPT}_z(D)/3$. Therefore, $B(o_{t-1}, d(o_{t-1}, f_t))$ is a solution for the 1-center problem on D with z outliers, which implies $\text{OPT}_z(D) \leqslant d(o_{t-1}, f_t)$. Hence, $\text{OPT}_z(D) \leqslant d(o_{t-1}, f_t) < \text{OPT}_z(D)$, which is a contradiction. ◀

▶ **Theorem 4.2.** *Let D be a set of n points in a metric space. Then there exists a coreset $\mathcal{C} \subset D$ of size at most $2z+2$ such that $\text{OPT}_z(D)/3 \leqslant \text{OPT}_z(\mathcal{C}) \leqslant \text{OPT}_z(D)$.*

Proof. Let $\mathcal{C}$ be the coreset returned by FindCoreset(D, z), and assume $|D| > 2z+2$ so that $\mathcal{C} = C_{2z+2}$. Since $C_{2z+2} \subseteq D$, then $\text{OPT}_z(C_{2z+2}) \leqslant \text{OPT}_z(D)$ trivially holds. To prove the other side of the inequality, suppose for a contradiction that $\text{OPT}_z(C) < \text{OPT}_z(D)/3$. Let $B(o, \text{OPT}_z(C_{2z+2}))$ be the optimal solution for the 1-center problem on C_{2z+2} with z outliers. Then, as $|C_{2z+2}| = 2z+2$ and at most z points are outliers, $B(o, \text{OPT}_z(C_{2z+2}))$ contains at least $z+2$ points. However, since we assume $\text{OPT}_z(C) < \text{OPT}_z(D)/3$, then $B(o, C_{2z+2})$ contains at most $z+1$ points by Lemma 4.1, which is a contradiction. ◀

A $(1+\varepsilon)$-coreset in the plane. The algorithm above works for any metric space, giving a 3-approximation. Now we explain that if D is a set of points in $\mathbb{R}^2$, we can improve the approximation ratio and obtain an ε-coreset, for any given $\varepsilon > 0$. To accomplish this, we add $O(z/\varepsilon)$ extra points to the coreset as follows. Let C_{2z+2} be the output of FindCoreset(D, z) and $B(o_{2z+2}, r_{2z+2})$ be an optimal solution for the 1-center problem on C_{2z+2} with z outliers. We partition the plane into $\ell = \lceil \frac{12\pi}{\varepsilon} \rceil$ cones $K_1, K_2, ..., K_\ell$ centered at o_{2z+2} with an opening angle of at most $\varepsilon/6$ each. Then, for each cone, we add $2z+2$ additional points to the coreset, namely the $z+1$ nearest points and the $z+1$ furthest points to o_{2k+2} from the points in D/C_{2k+2} that are located within that cone. Let A be the set of at most $(12\pi/\varepsilon) \cdot (2z+2)$ points selected in these cones, and define $\mathcal{C} := C_{2z+2} \cup A$. We will show that $\mathcal{C}$ is an ε-coreset.

▶ **Theorem 4.3.** *Let D be a set of points in $\mathbb{R}^2$. There exists an ε-coreset for the 1-center problem with z outliers for D of size $O(z/\varepsilon)$.*

Proof. Consider the set $\mathcal{C}$ defined above and let $B(\hat{o}, \hat{r})$ be an optimal solution for the 1-center problem on $\mathcal{C}$ with z outliers. It suffices to show that, for any point $q \in D \setminus \mathcal{C}$, we have that $\|\hat{o}q\| \leqslant (1+\varepsilon)\hat{r}$. Suppose for a contradiction that $\|\hat{o}q\| > (1+\varepsilon)\hat{r}$. Let $K_1, \ldots, K_\ell$ be the cones defined above, and recall that each cone has an angle of at most $\frac{\varepsilon}{6}$. As already mentioned, we place in our coreset the $z+1$ furthest and $z+1$ closest points to o_{2z+2}. Observe that $|\hat{o}o_{2z+2}| \leqslant 2\hat{r}$, since $B(o_{2z+2}, r_{2z+2})$ and $B(\hat{o}, \hat{r})$ have to intersect (otherwise there would be more than z outliers outside $B(\hat{o}, \hat{r})$). Note that this means that for any

point $p \in \partial B(\hat{o}, \hat{r})$, we have that $|o_{2z+2}p| \leqslant 3\hat{r}$. Let K_j denote the cone containing q. Since $q \notin A$, there exist $z+1$ points closer to o_{2z+2} than q and $z+1$ points further to o_{2z+2} than q. We denote the set of closer points by A_{close} and the set of further points by A_{far}. We have the following cases:

Case I: K_j does not intersect $B(\hat{o}, \hat{r})$. Then we clearly have a contradiction since $|K_j \cap A| \geqslant 2z+2$ while we are allowed at most z outliers.

Case II: K_j contains $B(\hat{o}, \hat{r})$. Then the opening angle of K_j is smallest when both of its sides are tangent to $B(\hat{o}, \hat{r})$. Let t denote one of the points of tangency. Then in the right triangle $\triangle o_{2z+2}ot$ we have $\frac{\varepsilon}{6} > \sin\left(\frac{\varepsilon}{6}\right) = \frac{\hat{r}}{|o_{2z+2}o|} \geqslant \frac{1}{2}$, which is a contradiction.

Case III: Otherwise, at least one of the sides of K_j intersects $B(\hat{o}, \hat{r})$. Let e_1 denote this side and e_2 denote the other side. Also, let e denote the half-line through $o_{2z+2}, \hat{o}$. Clearly e_1 will then also intersect $B(\hat{o}, (1+\varepsilon)\hat{r})$. We will now show the following claim.

▷ Claim 1. The side e_2 of K_j also has to intersect $B(\hat{o}, (1+\varepsilon)\hat{r})$.

Proof. Suppose for a contradiction that the claim is false. Then the opening angle of K_j is smallest when both e_1 and e_2 are tangent to $B(\hat{o}, \hat{r})$ and $B(\hat{o}, (1+\varepsilon)\hat{r})$, respectively. Let p_1, p_2 denote the points of tangency as in Figure 6(i). Let p denote the point where the line through $\hat{o}$ and p_1 intersects e_2. Then since p has to lie outside $B(\hat{o}, (1+\varepsilon)\hat{r})$, we have that $|p_1p| > \varepsilon\hat{r}$. Moreover, $|o_{2z+2}p_1| < |o_{2z+2}\hat{o}| < 2\hat{r}$ and so in the right triangle $\triangle o_{2z+2}p_1p$ we get $\tan\left(\frac{\varepsilon}{6}\right) = \frac{|p_1p|}{|o_{2z+2}p_1|} > \frac{\varepsilon\hat{r}}{2\hat{r}} = \frac{\varepsilon}{2}$, which is a contradiction, as $\tan\theta < 2\theta$ for small enough θ. ◁

Since both sides of K_j intersect $B(\hat{o}, (1+\varepsilon)\hat{r})$, we can partition K_j in two or three regions, depending on the location of o_{2z+2}. Namely, if o_{2z+2} lies inside $B(\hat{o}, (1+\varepsilon)\hat{r})$, then K_j can be partitioned in a region inside $B(\hat{o}, (1+\varepsilon)\hat{r})$ and a region outside $B(\hat{o}, (1+\varepsilon)\hat{r})$. If o_{2z+2} lies outside $B(\hat{o}, (1+\varepsilon)\hat{r})$, then K_j can be partitioned in three regions as in Figure 6(ii). Since the latter case is the most general, we will prove that one. The former case can be handled similarly. Without loss of generality, we will assume that e_2 lies above $\hat{o}$ and that the angle between e_2 and e is at least $\frac{\varepsilon}{12}$. The proof is slightly different, but essentially the same, depending on whether e is contained in K_j. To handle both at the same time, from now on we will let $e_3 \equiv e$ when e is contained in K_j and $e_3 \equiv e_1$ otherwise. Note that q has to lie either in region R_1 or R_3. We will now consider these two subcases.

Subcase I: $q \in R_1$. Observe that then the point x, where e_2 enters $B(\hat{o}, (1+\varepsilon)\hat{r})$ is the furthest q can be from o_{2z+2}. To derive a contradiction, it suffices to show that every point in A_{close} lies outside $B(\hat{o}, \hat{r})$. Since the point of $K_j \cap B(\hat{o}, \hat{r})$ closest to o_{2z+2}, is the point where e_3 enters $B(\hat{o}, \hat{r})$ (denoted by y), it suffices to show that $|o_{2z+2}x| < |o_{2z+2}y|$. We define $\phi = \angle xyo_{2z+2}$. Then, by the Law of Sines in the triangle $\triangle o_{2z+2}xy$ we get $\frac{\sin(\varepsilon/6)}{|xy|} = \frac{\sin(\phi)}{|o_{2z+2}x|}$. Therefore, $\sin\phi < \frac{\varepsilon/6}{\varepsilon r}\,|o_{2z+2}x| = \frac{|o_{2z+2}x|}{6\hat{r}}$.

Now notice that $|o_{2z+2}x| < |o_{2z+2}\hat{o}| < 2\hat{r}$. To see this, consider the tangent from o_{2z+2} to $B(\hat{o}, \hat{r})$ that lies above (o_{2z+2}, o) and let t be the point of tangency. Then x has to lie in the triangle $\triangle o_{2z+2}t\hat{o}$ and so $\angle o_{2z+2}x\hat{o} \geqslant \pi/2$. Therefore we get that $\sin\phi < \frac{1}{3}$, which gives us that $\phi < \frac{\pi}{6}$. Since we can assume $\varepsilon < 1$, we get that $\angle o_{2z+2}xy > \pi - \frac{\pi}{6} - \frac{\varepsilon}{6} > \pi/2$ and therefore we have that $|o_{2z+2}x| < |o_{2z+2}y|$.

Subcase II: $q \in R_3$. The approach is similar. The point $\hat{x}$ where e_2 exits $B(\hat{o}, (1+\varepsilon)\hat{r})$ is the closest q can be to o_{2z+2}. To derive a contradiction, it suffices to show that in that case every point in A_{far} lies outside $B(\hat{o}, \hat{r})$. Since the point of $K_j \cap B(\hat{o}, \hat{r})$ furthest from

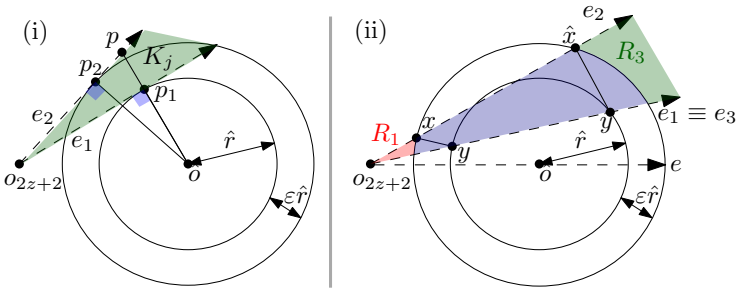


Figure 6 Illustrations for the proof of Theorem 4.3. Note that in (ii), e is not contained in the cone and therefore here we have $e_3 \equiv e_1$.

o_{2z+2}, is the point where e_3 exits $B(\hat{o}, \hat{r})$ – we denote this by $\hat{y}$ – it suffices to show that $|o_{2z+2}\hat{x}| > |o_{2z+2}\hat{y}|$. We define $\hat{\phi} = \angle\hat{y}\hat{x}o_{2z+2}$. Then, by the Law of Sines in the triangle $\triangle o_{2z+2}\hat{x}\hat{y}$ we have $\frac{\sin(\varepsilon/6)}{|\hat{x}\hat{y}|} = \frac{\sin(\hat{\phi})}{|o_{2z+2}\hat{y}|}$. Hence, $\sin\hat{\phi} < \frac{\varepsilon/6}{\varepsilon r}|o_{2z+2}\hat{y}| = \frac{|o_{2z+2}\hat{y}|}{6\hat{r}}$.

Now notice that $|o_{2z+2}\hat{y}| < 3\hat{r}$, as observed in the first paragraph of this proof. Therefore we get that $\sin\hat{\phi} < \frac{1}{2}$, which gives us that $\hat{\phi} < \frac{\pi}{6}$. Since we can assume $\varepsilon < 1$, we get that $\angle o_{2z+2}\hat{y}\hat{x} > \pi - \frac{\pi+1}{6} > \frac{\pi}{2}$ and therefore we have that $|o_{2z+2}\hat{x}| > |o_{2z+2}\hat{y}|$. This again gives a contradiction and concludes the proof. ◀

References

1 Pankaj K. Agarwal and Micha Sharir. Planar geometric location problems. *Algorithmica*, 11(2):185–195, 1994. doi:10.1007/BF01182774.

2 Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. *Discret. Comput. Geom.*, 56(4):836–859, 2016. doi:10.1007/s00454-016-9796-0.

3 Henk Alkema, Mark de Berg, Morteza Monemizadeh, and Leonidas Theocharous. TSP in a Simple Polygon. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2022.5.

4 Mihai Bâdoiu and Kenneth L. Clarkson. Smaller core-sets for balls. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003)*, pages 801–802, 2003.

5 Mihai Bâdoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proc. 34th Annual ACM Symposium on Theory of Computing (STOC 2002)*, pages 250–257, 2002. doi:10.1145/509907.509947.

6 Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. Solving k-center clustering (with outliers) in mapreduce and streaming, almost as accurately as sequentially. *Proc. VLDB Endow.*, 12(7):766–778, 2019. doi:10.14778/3317315.3317319.

7 Mark de Berg, Leyla Biabani, and Morteza Monemizadeh. *k*-center clustering with outliers in the MPC and streaming model. In *Proc. 37th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2023)*, pages 853–863, 2023. doi:10.1109/IPDPS54959.2023.00090.

8 Hristo Djidjev and Shankar M. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34(3):231–243, 1997. doi:10.1007/s002360050082.

9 Zvi Drezner. The *p*-centre problem-heuristic and optimal algorithms. *The Journal of the Operational Research Society*, 35(8):741–748, 1984. URL: http://www.jstor.org/stable/2581980.

10 Yunjun Gao and Baihua Zheng. Continuous obstructed nearest neighbor queries in spatial databases. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD 2009)*, pages 577–590, 2009. doi:10.1145/1559845.1559906.

11 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.

12 R. Z. Hwang, R. C. Chang, and Richard C. T. Lee. The searching over separators strategy to solve some NP-hard problems in subexponential time. *Algorithmica*, 9(4):398–423, 1993. doi:10.1007/BF01228511.

13 R. Z. Hwang, Richard C. T. Lee, and R. C. Chang. The slab dividing approach to solve the Euclidean p-center problem. *Algorithmica*, 9(1):1–22, 1993. doi:10.1007/BF01185335.

14 Dániel Marx and Michał Pilipczuk. Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams. *ACM Trans. Algorithms*, 18(2), 2022. doi:10.1145/3483425.

15 Nimrod Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. In *Proc. 23rd Annual Symposium on Foundations of Computer Science (FOCS 1982)*, pages 329–338, 1982. doi:10.1109/SFCS.1982.24.

16 Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984. doi:10.1137/0213014.

17 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986. doi:10.1016/0022-0000(86)90030-9.

18 Eunjin Oh, Sang Won Bae, and Hee-Kap Ahn. Computing a geodesic two-center of points in a simple polygon. *Computational Geometry*, 82:45–59, 2019. doi:10.1016/j.comgeo.2019.05.001.

19 Michael Ian Shamos and Dan Hoey. Closest-point problems. In *Proc. 16th Annual Symposium on Foundations of Computer Science (FOCS 1975)*, pages 151–162, 1975. doi:10.1109/SFCS.1975.8.

20 J. J. Sylvester. A question in the geometry of situation. *Quarterly Journal of Pure and Applied Mathematics*, 1857.

21 A.K.H. Tung, J. Hou, and Jiawei Han. Spatial clustering in the presence of obstacles. In *Proc. 17th International Conference on Data Engineering*, pages 359–367, 2001. doi:10.1109/ICDE.2001.914848.

22 Haitao Wang. On the planar two-center problem and circular hulls. *Discrete & Computational Geometry*, 68(4):1175–1226, 2022. doi:10.1007/s00454-021-00358-5.

23 Xin Wang and Howard J. HHamilton. Clustering spatial data in the presence of obstacles. *International Journal on Artificial Intelligence Tools*, 14:177–198, 2005. doi:10.1142/S0218213005002053.

24 Chenyi Xia, David Hsu, and Anthony K. H. Tung. A fast filter for obstructed nearest neighbor queries. In *Key Technologies for Data Management*, pages 203–215, 2004.

25 O.R. Zaiane and Chi-Hoon Lee. Clustering spatial data in the presence of obstacles: a density-based approach. In *Proc. International Database Engineering and Applications Symposium*, pages 214–223, 2002. doi:10.1109/IDEAS.2002.1029674.

Finding Diverse Minimum s-t Cuts

Mark de Berg
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Andrés López Martínez
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Frits Spieksma
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Abstract

Recently, many studies have been devoted to finding *diverse* solutions in classical combinatorial problems, such as Vertex Cover (Baste et al., IJCAI'20), Matching (Fomin et al., ISAAC'20) and Spanning Tree (Hanaka et al., AAAI'21). Finding diverse solutions is important in settings where the user is not able to specify all criteria of the desired solution. Motivated by an application in the field of *system identification*, we initiate the algorithmic study of k-Diverse Minimum s-t Cuts which, given a directed graph $G = (V, E)$, two specified vertices $s, t \in V$, and an integer $k > 0$, asks for a collection of k minimum s-t cuts in G that has maximum diversity. We investigate the complexity of the problem for two diversity measures for a collection of cuts: (i) the sum of all pairwise Hamming distances, and (ii) the cardinality of the union of cuts in the collection. We prove that k-Diverse Minimum s-t Cuts can be solved in strongly polynomial time for both diversity measures via *submodular function minimization*. We obtain this result by establishing a connection between ordered collections of minimum s-t cuts and the theory of distributive lattices. When restricted to finding only collections of mutually disjoint solutions, we provide a more practical algorithm that finds a maximum set of pairwise disjoint minimum s-t cuts. For graphs with small minimum s-t cut, it runs in the time of a single *max-flow* computation. These results stand in contrast to the problem of finding k diverse *global* minimum cuts – which is known to be NP-hard even for the disjoint case (Hanaka et al., AAAI'23) – and partially answer a long-standing open question of Wagner (Networks 1990) about improving the complexity of finding disjoint collections of minimum s-t cuts.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases S-T MinCut, Diversity, Lattice Theory, Submodular Function Minimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.24

Related Version *Full Version*: `https://arxiv.org/abs/2303.07290` [6]

Funding This research was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 945045, and by the NWO Gravitation project NETWORKS under grant no. 024.002.003.

Acknowledgements We thank Martin Frohn for bringing the theory of lattices to our attention, and for fruitful discussions on different stages of this work.

1 Introduction

The Minimum s-t Cut problem is a classic combinatorial optimization problem. Given a directed graph $G = (V, E)$ and two special vertices $s, t \in V$, the problem asks for a subset $S \subseteq E$ of minimum cardinality that separates vertices s and t, meaning that removing these edges from G ensures there is no path from s to t. Such a set is called a *minimum s-t cut* or *s-t mincut*, and it need not be unique. This problem has been studied extensively and has numerous practical and theoretical applications. Moreover, it is known to be solvable in polynomial time. Several variants and generalizations of the problem have been studied;

we mention the *global minimum cut* problem and the problem of *enumerating all minimum s-t cuts* in a graph. In this paper, we initiate the algorithmic study of computing *diverse* minimum *s-t* cuts. Concretely, we introduce the following optimization problem.

k-Diverse Minimum s-t Cuts (k-DMC). *Given are a directed graph $G = (V, E)$, vertices $s, t \in V$, and an integer $k > 0$. Let $\Gamma_G(s,t)$ be the set of minimum s-t cuts in G, and let U_k be the set of k-element multisets of $\Gamma_G(s,t)$. We want to find $C \in U_k$ such that $d(C) = \max_{S \in U_k} d(S)$, where $d : U_k \to \mathbb{N}$ is a measure of diversity.*

Informally, given a directed graph G, vertices s and t, and an integer k, we are interested in finding a collection of k *s-t* mincuts in G that are as different from each other as possible; that is, a collection having *maximum diversity*. Finding diverse solution sets is important in settings where the user is not able to specify all criteria of the desired solution. We mention the *synthesis problem* as an application of diverse minimum *s-t* cuts [27, 28].

To formally capture the notion of diversity of a collection of sets, several measures have been proposed (e.g., [30, 2, 15, 1, 13]). In this work, we choose two natural and general measures as our notions of diversity. Given a collection $(X_1, X_2, \ldots, X_k)$ of subsets of a set A (not necessarily distinct), we define $d_{\text{sum}}(X_1, \ldots, X_k) = \sum_{1 \le i < j \le k} |X_i \triangle X_j|$ and $d_{\text{cov}}(X_1, \ldots, X_k) = \left|\bigcup_{1 \le i \le k} X_i\right|$, where $X_i \triangle X_j = (X_i \cup X_j) \setminus (X_i \cap X_j)$ is the *symmetric difference* (or *Hamming distance*) of X_i and X_j. We call d_{sum} and d_{cov} the *pairwise-sum* and *coverage* diversity measures, respectively.

Our results. We investigate the complexity of the following two variants of k-Diverse Minimum s-t Cuts: (i) Sum k-Diverse Minimum s-t Cuts (Sum-k-DMC), and (ii) Cover k-Diverse Minimum s-t Cuts (Cov-k-DMC). These are the problems obtained when defining function d in k-DMC as diversity measures d_{sum} and d_{cov}, respectively. For a graph G, we use n to denote the number of nodes and m to denote the number of edges.

Contrary to the hardness of finding diverse *global* mincuts in a graph [13], we show that both Sum-k-DMC and Cov-k-DMC can be solved in polynomial time. We show this via a reduction to the *submodular function minimization* problem (SFM) on a *lattice*, which is known to be solvable in strongly polynomial time when the lattice is *distributive* [10, 16, 26].

▶ **Theorem 1.** *Sum-k-DMC and Cov-k-DMC can be solved in strongly polynomial time.*

At the core of this reduction is a generalization of an old result establishing a connection between minimum *s-t* cuts and distributive lattices [7]. As will be elaborated in Section 3, we obtain our results by showing that the pairwise-sum and coverage diversity measures (reformulated as minimization objectives) are *submodular* functions on the lattice L^* defined by *left-right ordered* collections of *s-t* mincuts and that this lattice is in fact *distributive*. Using the current fastest algorithm for SFM [17], together with an appropriate representation of the lattice L^*, we can obtain an algorithm that solves these problems in $O(k^5 n^5)$ time.

In Section 4, we obtain better time bounds for the special case of finding collections of *s-t* mincuts that are pairwise disjoint. Similar to SUM-k-DMC and COV-k-DMC, our approach exploits the partial order structure of *s-t* mincuts. We use this to efficiently solve the following optimization problem, which we call k-Disjoint Minimum *s-t* Cuts: given a graph $G = (V, E)$, vertices $s, t \in V$, and an integer $k \le k_{\max}$, find k pairwise disjoint *s-t* mincuts in G. Here, $k_{\max}$ denotes the maximum number of disjoint *s-t* mincuts in G. Our algorithm is significantly simpler than the previous best algorithm by Wagner [29], which runs in the time of a poly-logarithmic number of calls to any *min-cost flow* algorithm. Our algorithm takes $O(F(m,n) + m\lambda)$ time, where $F(m,n)$ is the time required by a unit-capacity

max-flow computation, and λ is the size of an s-t mincut in the graph. By plugging in the running time of the current fastest deterministic max-flow algorithms [21, 18], we obtain the following time bounds. When $\lambda \leq m^{1/3+o(1)}$, our algorithm improves upon the previous best runtime for this problem.

▶ **Theorem 2.** *k-Disjoint Minimum s-t Cuts can be solved in time $O(m^{4/3+o(1)} + m\lambda)$.*

Related Work. Many efforts have been devoted to finding diverse solutions in combinatorial problems. In their seminal paper [20], Kuo *et al.* were the first to explore this problem from a complexity-theoretic perspective. They showed that the basic problem of maximizing a distance norm over a set of elements is already NP-hard. Since then, the computational complexity of finding diverse solutions in many other combinatorial problems has been studied. For instance, diverse variants of Vertex Cover, Matching and Hitting Set have been shown to be NP-hard, even when considering simple diversity measures like the pairwise-sum of Hamming distances, or the minimum Hamming distance between sets. This has motivated the study of these and similar problems from the perspective of *fixed-parameter tractable* (FPT) algorithms [1, 2, 8]. Along the same line, Hanaka et al. [13] recently developed a framework to design approximation algorithms for diverse variants of combinatorial problems. On the positive side, diverse variants of other classic problems are known to be polynomially solvable, such as Spanning Tree [15], Shortest Path [14, 30], and Bipartite Matching [14], but not much is known about graph partitioning problems in light of diversity.

The problem of finding multiple minimum cuts has received considerable attention [13, 25, 29]. Picard and Queyranne [25] initiated the study of finding all minimum s-t cuts in a graph, showing that these can be enumerated efficiently. They observe that the closures of a naturally-defined poset over the vertices of the graph, correspond bijectively to minimum s-t cuts. An earlier work of Escalante [7] already introduced an equivalent poset for minimum s-t cuts, but contrary to Picard and Queyranne, no algorithmic implications were given. Nonetheless, Escalante shows that the set of s-t mincuts in a graph, together with this poset, defines a distributive lattice. Similar structural results for *stable matchings* and *circulations* have been shown to have algorithmic implications [11, 19], but as far as we know, the lattice structure of s-t mincuts has been seldomly exploited in the algorithmic literature.

Wagner [29] studied the problem of finding k pairwise-disjoint s-t cuts of *minimum total cost* in an edge-weighted graph. He showed that this problem can be solved in polynomial time by means of a reduction to a *transshipment* problem; where he raised the question of whether improved complexity bounds were possible by further exploiting the structure of the problem, as opposed to using a general purpose *min-cost flow* algorithm for solving the transshipment formulation. In sharp contrast, Hanaka et al. [13] recently established that the problem of finding k pairwise-disjoint *global* minimum cuts in a graph is NP-hard (for k part of the input). We are not aware of any algorithm for minimum s-t cuts that runs in polynomial time with theoretical guarantees on diversity.

2 Preliminaries

2.1 Distributive Lattices

In this paper, we use properties of distributive lattices. Here we introduce some basic concepts and results. For a more detailed introduction to lattice theory see e.g., [3, 5, 9].

A *partially ordered set (poset)* $P = (X, \preceq)$ is a ground set X together with a binary relation $\preceq$ on X that is reflexive, antisymmetric, and transitive. For a poset $P = (X, \preceq)$, an *ideal* is a set $U \subseteq X$ where $u \in U$ implies that $v \in U$ for all $v \preceq u$. We use $\mathcal{D}(P)$ to denote

the family of all ideals of P. When the binary operation $\preceq$ is clear from the context, we use the same notation for a poset and its ground set. We consider the standard representation of a poset P as a directed graph $G(P)$ containing a node for each element and edges from an element to its predecessors. In terms of $G(P) = (V, E)$, a subset W of V is an ideal if and only if there is no outgoing edge from W.

A *lattice* is a poset $L = (X, \preceq)$ in which any two elements $x, y \in X$ have a (unique) greatest lower bound, or *meet*, denoted by $x \wedge y$, as well as a (unique) least upper bound, or *join*, denoted by $x \vee y$. Hence, a lattice can also be identified by the tuple $(X, \vee, \wedge)$. A lattice L' is a *sublattice* of L if $L' \subseteq L$ and L' has the same meet and join operations as L. In this paper, we only consider *distributive lattices*, which are lattices whose meet and join operations satisfy distributivity; that is, $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ and $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$, for any $x, y, z \in L$. Note that a sublattice of a distributive lattice is also distributive.

Suppose we have a collection $L_1, \ldots, L_k$ of lattices $L_i = (X_i, \vee_i, \wedge_i)$ with $i \in \{1, ..., k\}$. The *(direct) product lattice* $L_1 \times \cdots \times L_k$ is a lattice with ground set $X = \{(x_1, \ldots, x_k) : x_i \in X_i\}$ and join $\vee$ and meet $\wedge$ acting component-wise; that is, $x \vee y = (x_1 \vee_1 y_1, \ldots, x_k \vee_k y_k)$ and $x \wedge y = (x_1 \wedge_1 y_1, \ldots, x_k \wedge_k y_k)$ for any $x, y \in X$. The lattice L^k is the product lattice of k copies of L and is called the kth *power* of L. If L is distributive, then L^k is also distributive.

A crucial notion in this work is that of *join-irreducibles*. An element x of a lattice L is called *join-irreducible* if it cannot be expressed as the join of two elements $y, z \in L$ with $y, z \neq x$. In a lattice, any element is equal to the join of all join-irreducible elements lower than or equal to it. The set of join-irreducible elements of L is denoted by $J(L)$. Note that $J(L)$ is a poset whose order is inherited from L. Due to Birkhoff's representation theorem every distributive lattice L is isomorphic to the lattice $\mathcal{D}(J(L))$ of ideals of its poset of join-irreducibles, with union and intersection as join and meet operations. Hence, a distributive lattice L can be uniquely recovered from its poset $J(L)$.

▶ **Theorem 3** (Birkhoff's Representation Theorem [3]). *Any distributive lattice L can be represented as the poset of its join-irreducibles $J(L)$, with the order induced from L.*

For a distributive lattice L, this implies that there is a *compact representation* of L as the directed graph $G(L)$ that characterizes its set of join-irreducibles. (The graph $G(L)$ is unique if we remove transitive edges.) This is useful when designing algorithms, as the size of $G(L)$ is $O(|J(L)|^2)$, while L can have as many as $2^{|J(L)|}$ elements.

2.2 Submodular Function Minimization

Let f be a real-valued function on a lattice $L = (X, \preceq)$. We say that f is *submodular* on L if $f(x \wedge y) + f(x \vee y) \leq f(x) + f(y)$, for all $x, y \in X$. If $-f$ is submodular in L, then we say that f is *supermodular* in L and just *modular* if f is both sub and supermodular. The *submodular function minimization* problem (SFM) on lattices is, given a submodular function f on L, to find an element $x \in L$ such that $f(x)$ is minimum. An important fact that we use in our work is that the sum of submodular functions is also submodular. Also, note that minimizing f is equivalent to maximizing $-f$.

Consider the special case of a lattice whose ground set $X \subseteq 2^U$ is a family of subsets of a set U, and meet and join are intersection and union of sets, respectively. It is known that any submodular function f on such a lattice can be minimized in polynomial time in $|U|$ [10, 16, 26], assuming that for any $Y \subseteq U$, the value of $f(Y)$ is given by an *evaluation oracle* that also runs in polynomial time in $|U|$. The current fastest algorithm for SFM on sets runs in $O(|U|^3 T_{\mathrm{EO}})$ time [17], where T_{EO} is the time required for one call to the evaluation oracle.

Due to Birkhoff's theorem, the seemingly more general case of SFM on distributive lattices can be reduced to SFM on sets (see [4, Sec. 3.1] for details). Hence, any polynomial-time algorithm for SFM on sets can be used to minimize a submodular function f defined on a distributive lattice L. An important remark is that the running time now depends on the size of the set $J(L)$ of join-irreducibles.

▶ **Theorem 4** ([24, Note 10.15] & [22, Thm.1]). *For any distributive lattice L, given by its poset of join-irreducibles $J(L)$, a submodular function $f : L \to \mathbb{R}$ can be minimized in polynomial time in $|J(L)|$, provided a polynomial time evaluation oracle for f.*

2.3 Minimum Cuts

Throughout this paper, we restrict our discussion to directed graphs. All our results can be extended to undirected graphs by means of well-known transformations. Likewise, we deal only with edge-cuts, although our approach can be easily adapted to vertex-cuts as well.

Let G be a directed graph with vertex set $V(G)$ and edge set $E(G)$. As usual, we define $n \coloneqq |V(G)|$ and $m \coloneqq |E(G)|$. Given a *source* $s \in V(G)$ and *target* $t \in V(G)$ in G, we call a subset $X \subset E(G)$ an *s-t cut* if the removal of X from the graph ensures that no path from s to t exists in $G \setminus X$. The size of a cut is the total number of edges it contains. If an s-t cut in G has smallest size $\lambda(G)$, we call it a minimum s-t cut, or an s-t mincut. Note that such a cut need not be unique (in fact, there can be exponentially many). To denote the set of all s-t mincuts of G, we use $\Gamma_G(s,t)$.

A (directed) path starting in a vertex u and ending in a vertex v is called a u-v path. By Menger's theorem, the cardinality of a minimum s-t cut in G is equal to the maximum number of internally edge-disjoint s-t paths in the graph. Let $\mathcal{P}_{s,t}(G)$ denote a maximum-sized set of edge-disjoint paths from s to t in G. Note that any minimum s-t cut in G contains exactly one edge from each path in $\mathcal{P}_{s,t}(G)$. For two distinct edges (resp. vertices) x and y in a u-v path p, we say that x is a path-predecessor of y in p and write $x \prec_p y$ if the path p meets x before y. We use this notation indistinctly for edges and vertices. It is easily seen that the relation $\prec_p$ extends uniquely to a non-strict partial order. We denote this partial order by $x \preceq_p y$. Consider now any subset $W \subseteq \Gamma_G(s,t)$ of s-t mincuts in G. We let $E(W) = \bigcup_{X \in W} X$. Two crucial notions in this work are those of *leftmost* and *rightmost* s-t mincuts. The *leftmost* s-t mincut in W consists of the set of edges $S_{\min}(W) \subseteq E(W)$ such that, for every path $p \in \mathcal{P}(s,t)$, there is no edge $e \in E(W)$ satisfying $e \prec_p f$ for any $f \in S_{\min}(W)$. Similarly for the *rightmost* s-t mincut $S_{\max}(W) \subseteq E(W)$. Note that both $S_{\min}(W)$ and $S_{\max}(W)$ are also s-t mincuts in G (see the proof of Proposition 5 in the full version of the paper [6]). When W consists of the entire set of s-t mincuts in G, we denote these extremal cuts by $S_{\min}(G)$ and $S_{\max}(G)$.

On the set of s-t cuts (not necessarily minimum), the following predecessor-successor relation defines a partial order: an s-t cut X is a predecessor of another s-t cut Y, denoted by $X \leq Y$, if every path from s to t in G meets an edge of X at or before an edge of Y. The set of s-t mincuts together with relation $\leq$ defines a distributive lattice L [7, 23]. Moreover, a compact representation $G(L)$ can be constructed from a maximum flow in linear time [25]. These two facts play a crucial role in the proof of our main result in the next section.

3 A Polynomial Time Algorithm for SUM-k-DMC and COV-k-DMC

This section is devoted to proving Theorem 1 by reducing SUM-k-DMC and COV-k-DMC to SMF on distributive lattices. First, we show that the domain of solutions of SUM-k-DMC and COV-k-DMC can be restricted to the set of k-tuples that satisfy a particular order, as opposed to the set of k-sized multisets of s-t mincuts (see Corollary 6 below). The reason

for doing so is that the structure provided by the former set can be exploited to assess the "modularity" of the total-sum and coverage objectives. Next, we introduce the notions of *left-right order* and *edge multiplicity*, which are needed throughout the section.

Consider a graph G with specified $s, t \in V(G)$, and let U^k be the set of all k-tuples over $\Gamma_G(s,t)$. An element $C \in U^k$ is a (ordered) collection or sequence $[X_1, \ldots, X_k]$ of cuts $X_i \in \Gamma_G(s,t)$, where i runs over the index set $\{1, \ldots, k\}$. We say that C is in *left-right order* if $X_i \leq X_j$ for all $i < j$. Let us denote by $U^k_{\text{lr}} \subseteq U^k$ the set of all k-tuples over $\Gamma_G(s,t)$ that are in left-right order. Then, for any two $C_1, C_2 \in U^k_{\text{lr}}$, with $C_1 = [X_1, \ldots, X_k]$, $C_2 = [Y_1, \ldots, Y_k]$, we say that C_1 is a *predecessor* of C_2 (and C_2 a *successor* of C_1) if $X_i \leq Y_i$ for all $i \in [k]$, and denote this by $C_1 \preceq C_2$. Now, consider again a collection $C \in U^k$. The set of edges $\bigcup_{X \in C} X$ is denoted by $E(C)$. We define the *multiplicity* of an edge $e \in E(G)$ with respect to (w.r.t.) C as the number of cuts in C that contain e and denote it by $\mu_e(C)$. We say that an edge $e \in E(C)$ is a *shared edge* if $\mu_e(C) \geq 2$. The set of shared edges in C is denoted by $E_{\text{shr}}(C)$. We make the following proposition, the proof of which is in the full version of the paper [6].

▶ **Proposition 5.** *For every $C \in U^k$ there exists $\hat{C} \in U^k_{\text{lr}}$ such that $\mu_e(C) = \mu_e(\hat{C})$ $\forall e \in E(G)$.*

In other words, given a k-tuple of s-t mincuts, there always exists a k-tuple on the same set of edges that is in left-right order; each edge occurring with the same multiplicity. Consider now the total-sum and the coverage diversity measures first introduced in Section 1. We can rewrite them directly in terms of the multiplicity of shared edges as

$$d_{\text{sum}}(C) = 2\left[\lambda(G)\binom{k}{2} - \sum\nolimits_{e \in E_{\text{shr}}(C)} \binom{\mu_e(C)}{2}\right] \quad \text{and} \tag{1}$$

$$d_{\text{cov}}(C) = k\lambda(G) - \sum\nolimits_{e \in E_{\text{shr}}(C)} (\mu_e(C) - 1)\,, \tag{2}$$

where terms outside the summations are constant terms. Then, combining eq. (1) (resp. (2)) with Proposition 5, we obtain the following corollary. (For simplicity, we state this only for the d_{sum} diversity measure, but an analogous claim holds for the d_{cov} measure.)

▶ **Corollary 6.** *Let $C \in U^k$ such that $d_{\text{sum}}(C) = \max_{S \in U^k} d_{\text{sum}}(S)$. Then there exists $C' \in U^k_{\text{lr}}$ such that $d_{\text{sum}}(C') = d_{\text{sum}}(C)$.*

This corollary tells us that in order to solve SUM-k-DMC (resp. COV-k-DMC) we do not need to optimize over the set U_k of k-element multisets of $\Gamma_G(s,t)$. Instead, we can look at the set $U^k_{\text{lr}} \subseteq U^k$ of k-tuples that are in left-right order. Moreover, it follows from Eqs. (1) and (2) that the problem of maximizing $d_{\text{sum}}(C)$ and $d_{\text{cov}}(C)$ is equivalent to minimizing

$$\hat{d}_{\text{sum}}(C) = \sum\nolimits_{e \in E_{\text{shr}}(C)} \binom{\mu_e(C)}{2}, \quad \text{and} \tag{3}$$

$$\hat{d}_{\text{cov}}(C) = \sum\nolimits_{e \in E_{\text{shr}}(C)} (\mu_e(C) - 1)\,, \tag{4}$$

respectively. In turn, the submodularity of $\hat{d}_{\text{sum}}(C)$ (resp. $\hat{d}_{\text{cov}}(C)$) implies the supermodularity of $d_{\text{sum}}(C)$ (resp. $d_{\text{cov}}(C)$) and vice versa. In the remaining of the section, we shall only focus on the minimization objectives $\hat{d}_{\text{sum}}$ and $\hat{d}_{\text{cov}}$.

We are now ready to show that both SUM-k-DMC and COV-k-DMC can be reduced to SFM. We first show that the poset $L^* = (U^k_{\text{lr}}, \preceq)$ is a distributive lattice (Section 3.1). Next we prove that the diversity measures $\hat{d}_{\text{sum}}$ and $\hat{d}_{\text{cov}}$ are submodular functions on L^* (Section 3.2). Lastly, we show that there is a compact representation of the lattice L^* and that it can be constructed in polynomial time, concluding with the proof of Theorem 1 (Section 3.3).

3.1 Proof of Distributivity

We use the following result of Escalante [7] (see also [12] or [23, Thm. 4]). Recall that $\leq$ denotes the predecessor-successor relation between two s-t mincuts.

▶ **Lemma 7.** *The set $\Gamma_G(s,t)$ of s-t mincuts of G together with the binary relation $\leq$ forms a distributive lattice L. For any two cuts $X, Y \in L$, the join and meet operations are given by $X \vee Y := S_{\max}(X \cup Y)$, and $X \wedge Y := S_{\min}(X \cup Y)$, respectively.*

By the definition of product lattice, we can extend this result to the relation $\preceq$ on the set U^k_{lr} of k-tuples of s-t mincuts that are in left-right order.

▶ **Lemma 8.** *The set U^k_{lr}, together with relation $\preceq$, defines a distributive lattice L^*. For any two elements $C_1 = [X_1, \ldots, X_k]$ and $C_2 = [Y_1, \ldots, Y_k]$ in L^*, the join and meet operations are given by $C_1 \vee C_2 = [S_{\max}(X_1 \cup Y_1), \ldots, S_{\max}(X_k \cup Y_k)]$ and $C_1 \wedge C_2 = [S_{\min}(X_1 \cup Y_1), \ldots, S_{\min}(X_k \cup Y_k)]$, respectively.*

Proof. This follows directly from Lemma 7 and the definition of product lattice (see Section 2.1). Let $L^k = (U^k, \preceq)$ be the kth power of the lattice $L = (\Gamma_G(s,t), \leq)$ of minimum s-t cuts, and let $L^* = (U^k_{\text{lr}}, \preceq)$ with $U^k_{\text{lr}} \subseteq U^k$ be the sublattice of left-right ordered k-tuples of minimum s-t cuts. We know from Section 2 that since L is distributive, then so is the power lattice L^k. Moreover, any sublattice of a distributive lattice is also distributive. Hence, it follows that the lattice L^* is also distributive. ◀

3.2 Proof of Submodularity

Now we prove that the functions $\hat{d}_{\text{sum}}$ and $\hat{d}_{\text{cov}}$ are submodular on the lattice L^*. We start with two lemmas that establish useful properties of the multiplicity function $\mu_e(C)$ on L^*. Due to space constraints, we defer the proofs to the full version of the paper [6].

▶ **Lemma 9.** *The multiplicity function $\mu_e : U^k_{\text{lr}} \to \mathbb{N}$ is modular on L^*.*

▶ **Lemma 10.** *For any two $C_1, C_2 \in L^*$ and $e \in E(C_1) \cup E(C_2)$, it holds that $\max(\mu_e(C_1 \vee C_2), \mu_e(C_1 \wedge C_2)) \leq \max(\mu_e(C_1), \mu_e(C_2))$.*

Lemma 10 plays an important role in the submodularity of $\hat{d}_{\text{sum}}$ and $\hat{d}_{\text{cov}}$. In contrast to Lemma 9, it does not hold on the kth power lattice of the distributive lattice L of Lemma 7.

Submodularity of $\hat{d}_{\text{sum}}$. Recall the definition of $\hat{d}_{\text{sum}}(C)$ in equation (3), and let $B_e : U^k_{\text{lr}} \to \mathbb{N}$ be the function defined by $B_e(C) = \binom{\mu_e(C)}{2}$. We can rewrite equation (3) as $\hat{d}_{\text{sum}}(C) = \sum_{e \in E_{\text{shr}}(C)} B_e(C)$. The following is an immediate consequence of Lemmas 9–10 and the convexity of $B_e(C)$.

▷ Claim 11. For any two $C_1, C_2 \in L^*$ and $e \in E(G)$, we have $B_e(C_1 \vee C_2) + B_e(C_1 \wedge C_2) \leq B_e(C_1) + B_e(C_2)$.

In other words, the function $B_e(C)$ is submodular in the lattice L^*. Now, recall that the sum of submodular functions is also submodular. Then, taking the sum of $B_e(C)$ over all edges $e \in E(G)$ results in a submodular function. From here, notice that $B_e(C) = 0$ for unshared edges; that is, when $\mu_e(C) < 2$. This means that such edges do not contribute to the sum. It follows that, for any two $C_1, C_2 \in L^*$, we have

$$\sum_{e \in E_{\text{shr}}(C_1 \vee C_2)} B_e(C_1 \vee C_2) + \sum_{e \in E_{\text{shr}}(C_1 \wedge C_2)} B_e(C_1 \wedge C_2) \leq \sum_{e \in E_{\text{shr}}(C_1)} B_e(C_1) + \sum_{e \in E_{\text{shr}}(C_2)} B_e(C_2).$$

Observe that each sum in the inequality corresponds to the definition of $\hat{d}_{sum}$ applied to the arguments $C_1 \vee C_2$, $C_1 \wedge C_2$, C_1 and C_2, respectively. Hence, by definition of submodularity, we obtain our desired result.

▶ **Theorem 12.** *The function $\hat{d}_{\text{sum}} : U_{\text{lr}}^k \to \mathbb{N}$ is submodular on the lattice L^*.*

Submodularity of $\hat{d}_{\text{cov}}$. Consider the function $F_e(C) : U_{\text{lr}}^k \to \mathbb{N}$ defined by $F_e(C) = \mu_e(C) - 1$. It is an immediate corollary of Lemma 9 that $F_e(C)$ is modular in L^*. Then, the sum $\sum_e F_e(C)$ taken over all edges $e \in E(G)$ is also modular. Notice that only shared edges in C contribute positively to the sum. The contribution of unshared edges is either neutral or negative. We can split this sum into two parts: the sum over shared edges $e \in E_{\text{shr}}(C)$, and the sum over $e \in E(G) \setminus E_{\text{shr}}(C)$. The latter can be further simplified to $|E(C)| - |E(G)|$ by observing that only the edges $e \in E(G) \setminus E(C)$ make a (negative) contribution. Therefore, we write

$$\sum\nolimits_{e \in E(G)} F_e(C) = \left(\sum\nolimits_{e \in E_{\text{shr}}(C)} (\mu_e(C) - 1) \right) + |E(C)| - |E(G)|. \tag{5}$$

We know $\sum_e F_e(C)$ to be a modular function on L^*, hence for any two $C_1, C_2 \in L^*$ we have $\sum_{e \in E(G)} F_e(C_1 \vee C_2) + \sum_{e \in E(G)} F_e(C_1 \wedge C_2) = \sum_{e \in E(G)} F_e(C_1) + \sum_{e \in E(G)} F_e(C_2)$, which, by equation (5), is equivalent to

$$\begin{aligned} &\left(\sum_{e \in E_{\text{shr}}(C_1 \vee C_2)} (\mu_e(C_1 \vee C_2) - 1) + \sum_{e \in E_{\text{shr}}(C_1 \wedge C_2)} (\mu_e(C_1 \wedge C_2) - 1) \right) \\ &\qquad + |E(C_1 \vee C_2)| + |E(C_1 \wedge C_2)| = \\ &= \left(\sum_{e \in E_{\text{shr}}(C_1)} (\mu_e(C_1) - 1) + \sum_{e \in E_{\text{shr}}(C_2)} (\mu_e(C_2) - 1) \right) + |E(C_1)| + |E(C_2)|. \end{aligned} \tag{6}$$

Now, from Lemmas 9 and 10, we have the following result, whose proof can be found in the full version [6].

▷ Claim 13. For any two $C_1, C_2 \in L^*$ we have $|E(C_1 \vee C_2)| + |E(C_1 \wedge C_2)| \geq |E(C_1)| + |E(C_2)|$.

Given Claim 13, it is clear that to satisfy equality in equation (6) it must be that:

$$\begin{aligned} &\sum_{e \in E_{\text{shr}}(C_1 \vee C_2)} (\mu_e(C_1 \vee C_2) - 1) + \sum_{e \in E_{\text{shr}}(C_1 \wedge C_2)} (\mu_e(C_1 \wedge C_2) - 1) \\ &\qquad \leq \sum_{e \in E_{\text{shr}}(C_1)} (\mu_e(C_1) - 1) + \sum_{e \in E_{\text{shr}}(C_2)} (\mu_e(C_2) - 1), \end{aligned}$$

from which the submodularity of $\hat{d}_{\text{cov}}$ immediately follows.

▶ **Theorem 14.** *The function $\hat{d}_{\text{cov}} : U_{\text{lr}}^k \to \mathbb{N}$ is submodular on the lattice L^*.*

3.3 Finding the Set of Join-Irreducibles

We now turn to the final part of the reduction to SFM. By Lemma 8, we know that the lattice L^* of left-right ordered collections of s-t mincuts is distributive. Moreover, it follows from Theorems 12 and 14 that the objective functions $\hat{d}_{\text{sum}}$ and $\hat{d}_{\text{cov}}$ are submodular in L^*. As discussed in Section 2.2, it now suffices to find an appropriate (compact) representation of L^* in the form of its poset of join-irreducibles $J(L^*)$.

Recall the distributive lattice L of s-t mincuts of a graph G defined in Lemma 7. The leftmost cut $S_{\min}(G)$ can be seen as the meet of all elements in L. In standard lattice notation, this smallest element is often denoted by $0_L := \bigvee_{x \in L} x$. We use the following result of Picard and Queyranne.

▶ **Lemma 15** ([25]). *Let L be the distributive lattice of s-t mincuts in a graph G, there is a compact representation $G(L)$ of L with the following properties:*

1. *The vertex set is $J(L) \cup 0_L$,*
2. *$|G(L)| \leq |V(G)|$,*
3. *Given G as input, $G(L)$ can be constructed in $F(n,m) + O(m)$ time.*

In other words, the set $J(L)$ is of size $O(n)$ and can be recovered from G in the time of a single max-flow computation. Moreover, each element of $J(L)$ corresponds to an s-t mincut in G. From this lemma, we obtain the following result for the poset $J(L^*)$, the proof of which is in the full version [6].

▶ **Lemma 16.** *The set of join-irreducibles of L^* is of size $O(kn)$ and is given by*

$$J(L^*) = \textstyle\bigcup_{i=1}^{k} J_i, \text{ where } J_i := \{(\underbrace{0_L, \ldots, 0_L}_{i-1 \text{ times}}, \underbrace{p, \ldots, p}_{k-i+1 \text{ times}}) : p \in J(L)\}.$$

Given Lemma 16, a compact representation of the lattice L^* can be obtained as the directed graph $G(L^*)$ that characterizes its poset of join-irreducibles $J(L^*)$ in polynomial time (since $|J(L^*)|$ is polynomial). It is also clear that the functions $\hat{d}_{\text{sum}}$ and $\hat{d}_{\text{cov}}$ can be computed in polynomial time. Then, by Theorem 4, the reduction to SFM is complete.

▶ **Theorem 1.** *Sum-k-DMC and Cov-k-DMC can be solved in strongly polynomial time.*

Due to space limitations, we refer the reader to the full version [6] for details on designing $O(k^5n^5)$-time algorithms for these problems.

4 A Simple Algorithm for Finding Disjoint Minimum s-t Cuts

In the previous section, we looked at the problem of finding the k most diverse minimum s-t cuts in a graph. Here, we consider a slightly different problem. Observe that for diversity measures d_{sum} and d_{cov}, the maximum diversity is achieved when the elements of a collection are all pairwise disjoint. Thus, it is natural to ask for a maximum cardinality collection of s-t mincuts that are pairwise disjoint; i.e., that are as diverse as possible. We call this problem Maximum Disjoint Minimum s-t Cuts (or Max-Disjoint MC for short).

Max-Disjoint MC. *Given a graph $G = (V, E)$ and vertices $s, t \in V(G)$, find a set $S \subseteq \Gamma_G(s,t)$ such that $X \cap Y = \emptyset$ for all $X, Y \in S$, and $|S|$ is as large as possible.*

Observe that a solution to Max-Disjoint MC immediately yields a solution to k-Disjoint Minimum s-t Cuts. In this section, we prove Theorem 2 by giving an algorithm for Max-Disjoint MC that runs in $O(F(m,n) + \lambda(G)m)$ time, where $F(m,n)$ is the time required by a max-flow computation. First, we look at a restricted case when the input graph can be decomposed into a collection of edge-disjoint s-t paths and (possibly) some additional edges – we refer to such a graph as an *s-t path graph* – and devise an algorithm that handles such graphs. Then, we use this algorithm as a subroutine to obtain an algorithm that makes no assumption about the structure of the input graph.

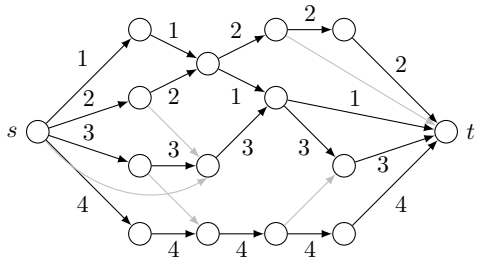


Figure 1 Example of an s-t path graph of height 4. Edges are labeled by integers corresponding to the path they belong to. Path edges are drawn in black and non-path edges in gray.

4.1 When the input is an s-t path graph

Let $H_{s,t}$ be a graph with designated vertices s and t. We call $H_{s,t}$ an *s-t path graph* (or *path graph* for short) if there is a collection P of edge-disjoint s-t paths such that P covers all vertices in $V(H_{s,t})$. The *height* of $H_{s,t}$, denoted by $\lambda(H_{s,t})$, is the maximum number of edge-disjoint s-t paths in the graph. For fixed P, we call the edges of $H_{s,t}$ in P *path edges* and edges of $H_{s,t}$ not in P *non-path edges*. Two vertices in $H_{s,t}$ are *path neighbors* if they are joined by a path edge, and *non-path neighbors* if they are joined (exclusively) by a non-path edge. See Figure 1 for an illustration.

Two remarks are in order. The first is that, by Menger's theorem, the size of a minimum s-t cut in an s-t path graph coincides with its height. The second remark is that, from a graph G, one can easily obtain a path graph $H_{s,t}$ of height $\lambda(G)$ by finding a maximum-sized set $\mathcal{P}_{s,t}$ of edge-disjoint s-t paths in G and letting $H_{s,t}$ be the induced subgraph of their union. Recall that, by Menger's theorem, a minimum s-t cut in G must contain exactly one edge from each path $p \in \mathcal{P}_{s,t}$. Thus, every minimum s-t cut of G is in $H_{s,t}$. However, the reverse is not always true. In the above construction, there could be multiple new minimum s-t cuts introduced in $H_{s,t}$ that arise from ignoring the reachability between vertices of $\mathcal{P}_{s,t}$ in G. We will come back to this issue when discussing the general case in Section 4.2.

The algorithm. The goal in this subsection is to find a maximum cardinality collection $\hat{C}$ of pairwise disjoint s-t mincuts in a path graph $H_{s,t}$. We now explain the main ideas behind the algorithm. Without loss of generality, assume that the underlying set of edge-disjoint s-t paths that define $H_{s,t}$ is of maximum cardinality.

Let X be an s-t mincut in $H_{s,t}$, and suppose we are interested in finding an s-t mincut Y disjoint from X such that $X < Y$. Consider any two edges $e = (u, u')$ and $f = (v, v')$ in X, and let $g = (w, w')$ be a path successor of f; that is $f \prec_p g$ with $p \in \mathcal{P}_{s,t}$. If there is a non-path edge $h = (u', z)$ such that $w' \leq z$, we say that h is *crossing* w.r.t. g, and that g is *invalid* w.r.t. X (see Figure 2 for an illustration). The notions of crossing and invalid edges provide the means to identify the edges that cannot possibly be contained in Y. Let $E_{\text{inv}}(X)$ denote the set of invalid edges w.r.t. X. We make the following observation.

▶ **Observation 17.** *Let $Y > X$. Then Y cannot contain an edge from $E_{\text{inv}}(X)$.*

Proof. For the sake of contradiction, suppose there exists an edge $g = (w, w')$ in $E_{\text{inv}}(X) \cap Y$. Consider the path $p_1 \in \mathcal{P}_{s,t}$, and let f be the predecessor of g on p_1 that is in X. Since $g \in E_{\text{inv}}(X)$, there is a crossing edge $h = (u', z)$ w.r.t. g. Let $p_2 \in \mathcal{P}_{s,t}$ be the path containing u', and let (u, u') be the edge of p_2 that is in X. Let p_3 be the s-t path that follows p_2 from s to u, then follows the crossing edge h, and then continues along p_1 to t. Since Y is an s-t

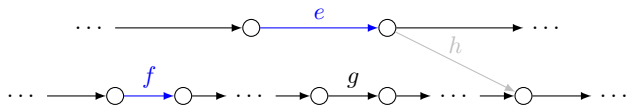


Figure 2 Example illustrating the notions of crossing and invalid edges for an s-t mincut X. Path and non-path edges are drawn in black and gray, respectively. Edges $e, f \in X$ are highlighted in blue. The edge g is invalid w.r.t. X since the edge h is crossing with respect to it.

cut it must contain an edge from this path. Since Y must contain exactly one edge from each path in $\mathcal{P}_{s,t}$, it cannot contain h. Moreover, Y already contains edge g from p_1. Then Y must contain an edge from the part of p_2 from s to u'. But this contradicts our assumption that $Y > X$. ◀

If we extend the definition of $E_{\text{inv}}(X)$ to also include all the edges that are path predecessors of edges in X, we obtain that, for any s-t path $p \in \mathcal{P}_{s,t}$, the set of invalid edges along p is a prefix of the path. As a result, if we can identify the (extended) set $E_{\text{inv}}(X)$, then we can restrict our search of cut Y to the set of valid edges $E_{\text{val}}(X) := E(H_{s,t}) \setminus E_{\text{inv}}(X)$. This motivates the following iterative algorithm for finding a pairwise disjoint collection of s-t mincuts: (1) Find the leftmost s-t mincut X in $H_{s,t}$, (2) identify the set $E_{\text{inv}}(X)$ and find the leftmost s-t mincut Y amongst $E_{\text{val}}(X)$, (3) set $X = Y$ and repeat step (2) until $E_{\text{val}}(X) \cap p = \emptyset$ for any one path $p \in \mathcal{P}_{s,t}$, and finally (4) output the union of identified cuts as the returned collection $\hat{C}$. Informally, notice that the s-t mincut identified at iteration i is a successor of the mincuts identified at iterations $j < i$. Hence, the returned collection will consist of left-right ordered and pairwise disjoint s-t mincuts. Moreover, picking the leftmost cut at each iteration prevents the set of invalid edges from growing unnecessarily large, which allows for more iterations and thus, a larger set returned. Next, we give a more formal description of the algorithm, the details of which are presented in Algorithm 1.

Algorithm 1 Obtain a Maximum Set of Disjoint Minimum s-t Cuts.

Input: Path graph $H_{s,t}$.
Output: A maximum set $\hat{C}$ of disjoint s-t mincuts in $H_{s,t}$.

Initialize collection $\hat{C} \leftarrow \emptyset$ and set $M \leftarrow \{s\}$.
while t is unmarked **do** ▷ Traverse the graph from left to right.
  while M is not empty **do** ▷ Marking step.
    for each vertex $v \in M$ **do**
      for each path $p \in \mathcal{P}_{s,t}$ **do** ▷ Identify invalid edges.
        Identify the rightmost neighbor $u \in p$ of v reachable by a non-path edge.
        if u is unmarked **then**
          Mark u and all (unmarked) vertices that are path-predecessors of u.
    Set M to the set of newly marked vertices.
  $X \leftarrow \bigcup\{(x,y) \in \mathcal{P}_{s,t} : x \text{ is marked, } y \text{ is unmarked}\}$. ▷ Cut-finding step.
  $\hat{C} \leftarrow \hat{C} \cup \{X\}$.
  for each $(x,y) \in X$ **do** ▷ Mark the head node of cut edges.
    Mark y.
  $M \leftarrow \bigcup_{(x,y)\in X} y$. ▷ Newly marked vertices.
Return $\hat{C}$.

The algorithm works by traversing the graph from left to right in iterations while marking the vertices it visits. Initially, all vertices are unmarked, except for s. Each iteration consists of two parts: a *marking*, and a *cut-finding* step. In the marking step (Lines 3-9), the algorithm identifies currently invalid edges by marking the non-path neighbors – and their path-predecessors – of currently marked vertices. (Observe that a path edge becomes invalid if both of its endpoints are marked.) In Algorithm 1, this is realized by a variable M that keeps track of the vertices that have just been marked as a consequence of the marking of vertices previously present in M. In the cut-finding step (Lines 10-14), the algorithm then finds the leftmost minimum s-t cut amongst valid path edges. Notice that, for each s-t path in $\mathcal{P}_{s,t}$, removing its first valid edge prevents s from reaching t via that path. This means that our leftmost cut of interest is the set of all path edges that have exactly one of their endpoints marked. Following the identification of this cut, the step concludes by marking the head vertices of the identified cut edges. Finally, the algorithm terminates when the target vertex t is visited and marked. See Figure 3 for an example execution of the algorithm.

▷ Claim 18. The complexity of Algorithm 1 on an m-edge, n-vertex path graph is $O(m \log n)$.

Due to space limitations, we defer the proof of Claim 18 to the full version [6].

Correctness of Algorithm 1. We note an important property of collections of s-t mincuts. (We use $d(C)$ to denote any of $d_{\text{sum}}(C)$ or $d_{\text{cov}}(C)$.)

▷ Claim 19. Let C be a left-right ordered collection of minimum s-t cuts in a graph G, the collection $\tilde{C}$ obtained by replacing $S_{\min}(\bigcup_{X \in C} X)$ (resp. $S_{\max}(\bigcup_{X \in C} X)$) with $S_{\min}(G)$ (resp. $S_{\max}(G)$) has cost $d(\tilde{C}) \leq d(C)$.

Proof. We prove this only for $S_{\min}(\cdot)$ as the proof for $S_{\max}(\cdot)$ is analogous. For simplicity, let us denote $S_{\min}(C) := S_{\min}(\bigcup_{X \in C} X)$. By definition, we know that no edge of $\bigcup_{X \in C} X$ lies to the left of $S_{min}(G)$. Then replacing $S_{\min}(C)$ with $S_{\min}(G)$ can only decrease the number of pairwise intersections previously present between $S_{\min}(C)$ and the cuts in $C \setminus S_{\min}(C)$. Notice that our measures of diversity only penalize edge intersections. Hence, the cost of collection $\tilde{C}$ cannot be greater than that of C. ◁

Now, consider an arbitrary collection of k edge-disjoint s-t mincuts in a path graph $H_{s,t}$. Corollary 6 implies that there also exists a collection of k edge-disjoint s-t mincuts in $H_{s,t}$ that is in left-right order. In particular, this is true for a collection of maximum cardinality $k_{\mathbf{max}}$. Together with Claim 19, this means that there always exists a collection $\hat{C}$ of edge-disjoint s-t mincuts in $H_{s,t}$ with the following properties:

- **I** $\hat{C}$ has size $k_{\mathbf{max}}$,
- **II** $\hat{C}$ is in left-right order,
- **III** $\hat{C}$ contains the leftmost s-t mincut of $H_{s,t}$, and
- **IV** The set $S_{\max}(\hat{C}) \cap S_{\max}(H_{s,t})$ is not empty.

We devote the rest of the subsection to proving the following lemma, which serves to prove the correctness of Algorithm 1.

▶ **Lemma 20.** *Algorithm 1 returns a collection of edge-disjoint minimum s-t cuts that satisfies Properties I–IV.*

Let $\hat{C}$ denote the solution returned by the algorithm. First, we show that $\hat{C}$ contains only disjoint cuts. This follows from the fact that a cut can only be found amongst valid edges at any given iteration, and once an edge has been included in a cut, it becomes invalid

at every subsequent iteration. Similarly, Properties II and III are consequences of the notion of invalid edges. We start by proving the latter. Let X_1 denote the leftmost cut in $\hat{C}$. For the sake of contradiction, assume there is a minimum s-t cut Y such that $e \prec_p f$. Here, $e \in Y$, $f \in X_1$ and *w.l.o.g.* p is an s-t path from any arbitrary maximum collection of s-t paths in $H_{s,t}$. For the algorithm to pick edge $f = (u, u')$ as part of X_1 it must be that vertex u is marked and u' is not. We know that the predecessors of marked vertices must also be marked. Hence we know that both endpoints of edge e are marked. But by definition, this means that edge e is invalid, and cannot be in a minimum s-t cut. This gives us the necessary contradiction, and X_1 must be the leftmost cut in the graph.

We continue with Property II. This property follows from the fact that, at any given iteration, the posets of invalid path-edges on each path of $H_{s,t}$ are ideals of the set of path edges. This means that the edges in the cut found by the algorithm at iteration i are all path predecessors of an edge in the cut found at iteration $i+1$. Carrying on with Property IV, we prove that it follows from the fact that the algorithm terminates when the target node t is marked. Suppose, for the sake of contradiction, that the cuts $S_{\max}(\hat{C})$ and $S_{\max}(H_{s,t})$ do not intersect. Then, given that $S_{\max}(\hat{C})$ is the last cut found by our algorithm, to mark node t there must exist a non-path edge connecting the endpoint v of some edge $e = (u, v) \in S_{\max}(\hat{C})$ to t. But this implies that no path-successor of edge e can be in an s-t mincut, which makes e the rightmost edge on its path that belongs to an s-t mincut. Therefore, e must also be contained in $S_{\max}(H_{s,t})$, a contradiction.

It only remains to show Property I, which states that the collection $\hat{C}$ is of maximum cardinality $k_{\mathbf{max}}$. For this, we make the following claim, whose proof is analogous to the proof of Property III. Let $\hat{C}_i$ be the collection of s-t mincuts maintained by the algorithm at the end of iteration i.

▷ Claim 21. Consider set $\hat{C}_{i-1}$ and let X_i be the minimum s-t cut found by the algorithm at iteration i. Then, there is no minimum s-t cut Y such that: (i) Y is disjoint from each $X \in \hat{C}_{i-1}$, and (ii) Y contains an edge that is a path predecessor of an edge of X_i.

In other words, as the algorithm makes progress, no minimum s-t cut – that is disjoint from the ones found so far by the algorithm – has edges to the left of the minimum s-t cut found by the algorithm at the present iteration. Next, we show that this implies the maximality of the size of the solution returned by the algorithm.

Let $C_{\max}$ be a maximum-sized collection of s-t mincuts in the graph. Without loss of generality, assume that $C_{\max}$ is in left-right order (otherwise, by Corollary 6 we can always obtain an equivalent collection that is left-right ordered) and that $S_{\min}(H_{s,t}) \in C_{\max}$ and $S_{\max}(H_{s,t}) \in C_{\max}$. For the sake of contradiction, suppose that the collection $\hat{C}$ returned by our algorithm is of cardinality $|\hat{C}| = \ell < k_{\max}$.

▶ **Observation 22.** *There exists at least one minimum s-t cut $Y \in C_{\max}$ such that $X_i < Y$ and Y contains at least one edge that is a path predecessor of an edge in X_{i+1}, with X_i and X_{i+1} two consecutive cuts in $\hat{C}$.*

Proof. Let $C_{\max} = \{Y_1, Y_2, \ldots, Y_{k_{\max}}\}$, where $Y_1 = S_{\min}(H_{s,t})$ and $Y_{k_{\max}} = S_{\max}(H_{s,t})$, and let $\hat{C} = \{X_1, \ldots, X_\ell\}$. We know by Property III that $X_1 = S_{\min}(H_{s,t})$. Hence, there is always an s-t mincut in $C_{\max}$ that is a strict successor of a cut in $\hat{C}$, namely $Y_2 > X_1$. For the sake of contradiction, suppose that the observation is false. Then, every cut $Y \in C_{\max}$ that is a strict successor of a cut $X_i \in \hat{C}$ is also a (not necessarily strict) successor of the cut $X_{i+1} \in \hat{C}$, for $i \in \{1, \ldots, \ell - 1\}$. Let this be true for the first $\ell - 1$ cuts of $\hat{C}$. Then, the last $k_{\max} - \ell$ cuts of $C_{\max}$ must be disjoint from the first $\ell - 1$ cuts of $\hat{C}$. The last cut X_ℓ of $\hat{C}$

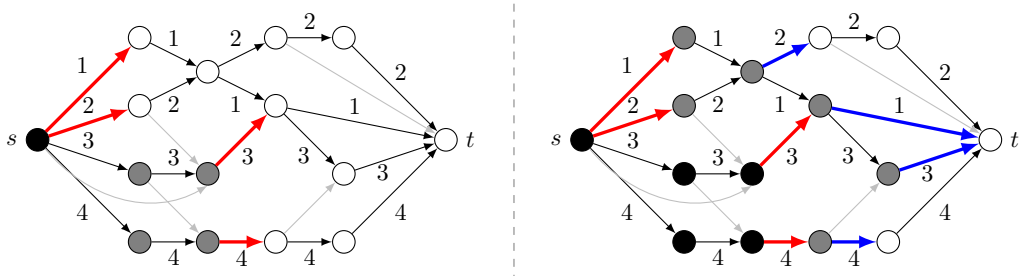


Figure 3 Example illustrating the first two iterations of Algorithm 1 on a path graph of height 4. The black- and gray-shaded vertices represent vertices marked at the previous and current iterations, respectively. The red edges correspond to the s-t mincut found at the end of the first (left) iteration. Similarly, the blue edges correspond to the s-t mincut found at the second (right) iteration.

must then be located in or before the gap between the first ℓ cuts in $C_{\max}$ and its remaining $k - \ell$ cuts. But we know by Property IV that $X_\ell \cap Y_{k_{\max}} \neq \emptyset$, which gives the necessary contradiction. ◀

Observation 22 stands in contrast with Claim 21, which states that such a cut Y cannot exist. Hence, we obtain a contradiction, and the collection $\hat{C}$ returned by the algorithm must be of maximum cardinality. This completes the proof of Lemma 20.

4.2 Handling the general case

We now consider Max-Disjoint MC in general graphs. Recall from the previous subsection that, from a graph G, one can construct a path graph $H_{s,t}$ such that every minimum s-t cut in G is also a minimum s-t cut in $H_{s,t}$. We could propose to use Algorithm 1 in $H_{s,t}$ to solve Max-Disjoint MC in G. But, as we argued previously, the path graph $H_{s,t}$ may not have the same set of s-t mincuts as G. We can, however, solve this challenge by augmenting $H_{s,t}$ with edges such that its minimum s-t cuts correspond bijectively to those in G.

▶ **Definition 23.** An *augmented s-t path graph* of G is a path graph $H_{s,t}(G)$ of height $\lambda(G)$, with additional non-path edges between any two vertices $u, v \in V(H_{s,t}(G))$ such that v is reachable from u in G by a path whose internal vertices are exclusively in $V(G) \setminus V(H_{s,t}(G))$.

In view of this definition, the following claim and lemma serve as the correctness and complexity proofs of the proposed algorithm for the general case.

▷ Claim 24. An augmented s-t path graph of G has the same set of s-t mincuts as G.

Proof. By Menger's theorem, we know that a minimum s-t cut in G must contain exactly one edge from each path in $\mathcal{P}_{s,t}(G)$, where $|\mathcal{P}_{s,t}(G)| = |\lambda(G)|$. W.l.o.g., let $H_{s,t}(G)$ be the augmented s-t path graph of G such that each path $p \in \mathcal{P}_{s,t}(G)$ is also present in $H_{s,t}(G)$. We now show that a minimum s-t cut in G is also present in $H_{s,t}(G)$. The argument in the other direction is similar and is thus omitted.

Consider an arbitrary minimum s-t cut X in G. For the sake of contradiction, assume that X is not a minimum s-t cut in $H_{s,t}(G)$. Then, after removing every edge of X in $H_{s,t}(G)$, there is still at least one s-t path left in the graph. Such a path must contain an edge (u, v) such that $u \leq w$ and $w' \leq v$, where w and w' are the tail and head nodes of two (not necessarily distinct) edges in X, respectively. By definition of $H_{s,t}(G)$, there is a path from u to v in G that does not use edges in $\mathcal{P}_{s,t}(G)$. But then removing the edges of X in G still leaves an s-t path in the graph. Thus X cannot be an s-t cut, and we reach our contradiction. ◁

▶ **Lemma 25.** *An augmented s-t path graph H of a graph G can be constructed in time $O(F(m,n) + m\lambda(G))$, where $F(m,n)$ is the time required by a max-flow computation.*

Proof. The idea of the algorithm is rather simple. First, we find a maximum cardinality collection of edge-disjoint s-t paths in G and take their union to construct a "skeleton" graph H. Then, we augment the graph by drawing an edge between two path vertices $u, v \in H$ if v is reachable from u in G by using exclusively non-path vertices. By definition, the resulting graph is an augmented s-t path graph of G.

Now we look into the algorithm's implementation and analyze its running time. It is folklore knowledge that the problem of finding a maximum-sized collection of edge-disjoint s-t paths in a graph with n vertices and m edges can be formulated as a maximum flow problem. Hence, the first step of the algorithm can be performed in $F(m,n)$ time. Let $\mathcal{P}_{s,t}(G)$ denote such found collection of s-t paths.

The second step of the algorithm could be computed in $O(mn)$ time by means of an *all-pairs reachability* algorithm. Notice, however, that for a path vertex v all we require for a correct execution of Algorithm 1 is knowledge of the rightmost vertices it can reach on each of the $\lambda(G)$ paths (Line 6 of Algorithm 1). Hence, we do not need to draw every edge between every pair of reachable path vertices, only $\lambda(G)$ edges per vertex suffice. This can be achieved in $O(m\lambda(G))$ time as follows.

In the original graph, first equip each vertex $u \in V(G)$ with a set of $\lambda(G)$ variables $R(p,u)$, one for each path $p \in \mathcal{P}_{s,t}(G)$. These variables will be used to store the rightmost vertex $v \in p$ that is reachable from u. Next, consider a path $p \in \mathcal{P}_{s,t}(G)$ represented as a sequence $[v_1, v_2, \ldots, v_p]$ of internal vertices (i.e., with s and t removed). For each vertex $v \in p$, in descending order, execute the following procedure `propagate(`v`, `p`)`: Find the set $N(v)$ of incoming neighbors of v in G and, for each $w \in N(v)$ if $R(p,w)$ has not been set, let $R(p,w) = v$ and mark w as visited. Then, for each node $w \in N(v)$, if w is an unvisited non-path vertex, execute `propagate(`w`, `p`)`; otherwise, do nothing. Notice that, since we iterate from the rightmost vertex in p, any node u such that $R(u,p) = v_i$ cannot change its value when executing `propagate(`v_j`)` with $j < i$. In other words, each vertex only stores information about the rightmost vertex it can reach in p. Complexity-wise, every vertex v in G will be operated upon at most $deg(v)$ times. Hence, starting from an unmarked graph, a call to `propagate(`v`, `p`)` takes $O(m)$ time. Now, we want to execute the above for each path $p \in \mathcal{P}_{s,t}(G)$ (unmarking all vertices before the start of each iteration). This then gives us our claimed complexity of $O(m\lambda(G))$. The claim follows from combining the running time of both steps of the algorithm. ◀

The following is an immediate consequence of Lemma 25 and Claim 18.

▶ **Corollary 26.** *There is an algorithm that, given a graph G and two specified vertices $s, t \in V(G)$, in $O(F(m,n) + m\lambda(G))$ time finds a collection of maximum cardinality of pairwise disjoint s-t mincuts in G.*

By replacing $F(m,n)$ in Corollary 26 with the running time of the current best algorithms for finding a maximum flow [21, 18], we obtain the desired running time of Theorem 2.

5 Concluding remarks

We showed that k-DMC can be solved efficiently when considering two natural measures for the diversity of a set of solutions. There exist, however, other sensible measures of diversity. One that often arises in literature is that of maximizing the minimum pairwise Hamming distance of a solution set. The complexity of k-DMC when considering this objective is still open. It is not immediately clear how to apply our ordering results to this variant.

For the special case of k-Disjoint Minimum s-t Cuts, we showed that faster algorithms exist when compared to solving k-DMC on the total-sum and coverage diversity measures. It is thus natural to ask whether there are faster algorithms for Sum-k-DMC and Cov-k-DMC (or other variants of k-DMC) that do not require the sophisticated framework of submodular function minimization. In this work, we relied on the algebraic structure of the problem to obtain a polynomial time algorithm. We believe it is an interesting research direction to assess whether the notion of diversity in other combinatorial problems leads to similar structures, which could then be exploited for developing efficient algorithms.

References

1 Julien Baste, Michael R. Fellows, Lars Jaffke, Tomáš Masařík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A. Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. *Artificial Intelligence*, 303:103644, 2022. doi:10.1016/j.artint.2021.103644.

2 Julien Baste, Lars Jaffke, Tomáš Masařík, Geevarghese Philip, and Günter Rote. Fpt algorithms for diverse collections of hitting sets. *Algorithms*, 12(12):254, 2019.

3 Garrett Birkhoff. Rings of sets. *Duke Mathematical Journal*, 3(3):443–454, 1937.

4 Mohammadreza Bolandnazar, Woonghee Tim Huh, S Thomas McCormick, and Kazuo Murota. A note on "order-based cost optimization in assemble-to-order systems". *University of Tokyo (February, Techical report)*, 2015.

5 Brian A Davey and Hilary A Priestley. *Introduction to lattices and order*. Cambridge University Press, 2002.

6 Mark de Berg, Andrés López Martínez, and Frits Spieksma. Finding diverse minimum s-t cuts, 2023. arXiv:2303.07290.

7 Fernando Escalante. Schnittverbände in graphen. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 38, pages 199–220. Springer, 1972.

8 Fedor V. Fomin, Petr A. Golovach, Lars Jaffke, Geevarghese Philip, and Danil Sagunov. Diverse Pairs of Matchings. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:12, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2020.26.

9 George Gratzer. *Lattice theory: First concepts and distributive lattices*. Courier Corporation, 2009.

10 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2. Springer Science & Business Media, 2012.

11 D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. Foundations of computing. MIT Press, 1989.

12 R Halin. Lattices related to separation in graphs. In *Finite and Infinite Combinatorics in Sets and Logic*, pages 153–167. Springer, 1993.

13 Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, Yusuke Kobayashi, Kazuhiro Kurita, and Yota Otachi. A framework to design approximation algorithms for finding diverse solutions in combinatorial problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 3968–3976, 2023.

14 Tesshu Hanaka, Yasuaki Kobayashi, Kazuhiro Kurita, See Woo Lee, and Yota Otachi. Computing diverse shortest paths efficiently: A theoretical and experimental study. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4):3758–3766, June 2022. doi:10.1609/aaai.v36i4.20290.

15 Tesshu Hanaka, Yasuaki Kobayashi, Kazuhiro Kurita, and Yota Otachi. Finding diverse trees, paths, and more. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3778–3786, 2021.

16 Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM (JACM)*, 48(4):761–777, 2001.

17 Haotian Jiang. Minimizing convex functions with integral minimizers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 976–985. SIAM, 2021.

18 Tarun Kathuria. A potential reduction inspired algorithm for exact max flow in almost $\widetilde{O}(m^{4/3})$ time, 2020. `arXiv:2009.03260`.

19 Samir Khuller, Joseph Naor, and Philip Klein. The lattice structure of flow in planar graphs. *SIAM Journal on Discrete Mathematics*, 6(3):477–490, 1993.

20 Ching-Chung Kuo, Fred Glover, and Krishna S Dhir. Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences*, 24(6):1171–1185, 1993.

21 Yang P. Liu and Aaron Sidford. Faster divergence maximization for faster maximum flow, 2020. `arXiv:2003.08929`.

22 George Markowsky. An overview of the poset of irreducibles. *Combinatorial And Computational Mathematics*, pages 162–177, 2001.

23 Bernd Meyer. On the lattices of cutsets in finite graphs. *European Journal of Combinatorics*, 3(2):153–157, 1982.

24 Kazuo Murota. *Discrete Convex Analysis*. Society for Industrial and Applied Mathematics, 2003. `doi:10.1137/1.9780898718508`.

25 Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *Mathematical Programming Studies*, 13:8–16, 1980.

26 Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000.

27 Shengling Shi, Xiaodong Cheng, and Paul M.J. Van den Hof. Generic identifiability of subnetworks in a linear dynamic network: The full measurement case. *Automatica*, 137:110093, 2022. `doi:10.1016/j.automatica.2021.110093`.

28 Shengling Shi, Xiaodong Cheng, and Paul M.J. Van den Hof. Personal communication, October 2021.

29 Donald K Wagner. Disjoint (s, t)-cuts in a network. *Networks*, 20(4):361–371, 1990.

30 Si-Qing Zheng, Bing Yang, Mei Yang, and Jianping Wang. Finding minimum-cost paths with minimum sharability. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*, pages 1532–1540. IEEE, 2007.

Efficient Algorithms for Euclidean Steiner Minimal Tree on Near-Convex Terminal Sets

Anubhav Dhar
Indian Institute of Technology Kharagpur, India

Soumita Hait
Indian Institute of Technology Kharagpur, India

Sudeshna Kolay
Indian Institute of Technology Kharagpur, India

Abstract

The Euclidean Steiner Minimal Tree problem takes as input a set $\mathcal{P}$ of points in the Euclidean plane and finds the minimum length network interconnecting all the points of $\mathcal{P}$. In this paper, in continuation to the works of [5] and [15], we study Euclidean Steiner Minimal Tree when $\mathcal{P}$ is formed by the vertices of a pair of regular, concentric and parallel n-gons.

We restrict our attention to the cases where the two polygons are not very close to each other. In such cases, we show that Euclidean Steiner Minimal Tree is polynomial-time solvable, and we describe an explicit structure of a Euclidean Steiner minimal tree for $\mathcal{P}$.

We also consider point sets $\mathcal{P}$ of size n where the number of input points not on the convex hull of $\mathcal{P}$ is $f(n) \leq n$. We give an exact algorithm with running time $2^{\mathcal{O}(f(n)\log n)}$ for such input point sets $\mathcal{P}$. Note that when $f(n) = \mathcal{O}(\frac{n}{\log n})$, our algorithm runs in single-exponential time, and when $f(n) = o(n)$ the running time is $2^{o(n \log n)}$ which is better than the known algorithm in [9].

We know that no FPTAS exists for Euclidean Steiner Minimal Tree unless P = NP [6]. On the other hand FPTASes exist for Euclidean Steiner Minimal Tree on convex point sets [14]. In this paper, we show that if the number of input points in $\mathcal{P}$ not belonging to the convex hull of $\mathcal{P}$ is $\mathcal{O}(\log n)$, then an FPTAS exists for Euclidean Steiner Minimal Tree. In contrast, we show that for any $\epsilon \in (0, 1]$, when there are $\Omega(n^{\epsilon})$ points not belonging to the convex hull of the input set, then no FPTAS can exist for Euclidean Steiner Minimal Tree unless P = NP.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Steiner minimal tree, Euclidean Geometry, Almost Convex point sets, FPTAS, strong NP-completeness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.25

Related Version *Full Version*: https://arxiv.org/abs/2307.00254

1 Introduction

The Euclidean Steiner Minimal Tree problem asks for a network of minimum total length interconnecting a given finite set $\mathcal{P}$ of n points in the Euclidean plane. Formally, we define the problem as follows, taken from [2]:

> Euclidean Steiner Minimal Tree
> **Input:** A set $\mathcal{P}$ of n points in the Euclidean plane
> **Question:** Find a connected plane graph $\mathcal{T}$ such that $\mathcal{P}$ is a subset of the vertex set $V(\mathcal{T})$, and for the edge set $E(\mathcal{T})$, $\Sigma_{e \in E(\mathcal{T})}\overline{e}$ is minimized over all connected plane graphs with $\mathcal{P}$ as a vertex subset.

Note that the metric being considered is the Euclidean metric, and for any edge $e \in E(\mathcal{T})$, $\overline{e}$ denotes the Euclidean length of the edge. Here, the input set $\mathcal{P}$ of points is often called a set of *terminals*, the points in $\mathcal{S} = V(\mathcal{T}) \setminus \mathcal{P}$ are called *Steiner points*. A solution graph $\mathcal{T}$ is referred to as a *Euclidean Steiner minimal tree*, or simply an SMT.

The Euclidean Steiner Minimal Tree problem is a classic problem in the field of Computational Geometry. The origin of the problem dates back to Fermat (1601-1665) who proposed the problem of finding a point in the plane such that the sum of its distance from three given points is minimized. This is equivalent to finding the location of the Steiner point when given three terminals as input. Torricelli proposed a geometric solution to this special case of 3 terminal points. The idea was to construct equilateral triangles outside on all three sides of the triangle formed by the terminals, and draw their circumcircles. The three circles meet at a single point, which is our required Steiner point. This point came to be known as the *Torricelli point*. When one of the angles in the triangle is at least 120°, the minimizing point coincides with the obtuse angle vertex of the triangle. In this case, the Torricelli point lies outside the triangle and no longer minimizes the sum of distances from the vertices. However, when vertices of polygons with more than 3 sides are considered as a set of terminals, a solution to the Fermat problem does not in general lead to a solution to the Euclidean Steiner Minimal Tree problem. For a more detailed survey on the history of the problem, please refer to [2, 9]. For convenience, we refer to the Euclidean Steiner Minimal Tree problem as ESMT.

ESMT is NP-hard. In [6], Garey et al. prove a discrete version of the problem (Discrete ESMT) to be strongly NP-complete via a reduction from the Exact Cover by 3-Sets (X3C) problem. Although it is not known if the ESMT problem is in NP, it is at least as hard as any NP-complete problem. So, we do not expect a polynomial time algorithm for it. A recursive method using only Euclidean constructions was given by Melzak in [11] for constructing all the Steiner minimal trees for any set of n points in the plane by constructing full Steiner trees of subsets of the points. Full Steiner trees are interconnecting trees having the maximum number of newly introduced points (Steiner points) where all internal junctions are of degree 3. Hwang improved the running time of Melzak's original exponential algorithm for full Steiner tree construction to linear time in [8]. Using this, we can construct an Euclidean Steiner minimal tree in $2^{\mathcal{O}(n \log n)}$ time for any set of n points. This was the first algorithm for Euclidean Steiner Minimal Tree. The problem is known to be NP-hard even if all the terminals lie on two parallel straight lines, or on a bent line segment where the bend has an angle of less than 120° [13]. Since the above sets of terminals all lie on the boundary of a convex polygon (or, are in convex position), this shows that ESMT is NP-hard when restricted to a set of points that are in weakly convex position.

Although the ESMT problem is NP-hard, there are certain arrangements of points in the plane for which the Euclidean Steiner minimal tree can be computed efficiently, say in polynomial time. One such arrangement is placing the points on the vertices of a regular polygon. This case was solved by Du et al. [5]. Their work gives exact topologies of the Euclidean Steiner minimal trees. Weng et al. [15] generalized the problem by incorporating the centre point of the regular polygon as part of the terminal set, along with the vertices. This case was also found to be polynomial time solvable.

Tractability in the form of approximation algorithms for ESMT has been extensively studied. It was proved in [6] that a fully polynomial time approximation scheme (FPTAS) cannot exist for this problem unless P = NP. However, we do have an FPTAS when the terminals are in convex position [14]. Arora's celebrated polynomial time approximation scheme (PTAS) for the ESMT and other related problems is described in [1]. Around the same time, Rao and Smith gave an efficient polynomial time approximation scheme (EPTAS) in [12]. In recent years, an EPTAS with an improved running time was designed by Kisfaludi-Bak et al. [10].

Our Results

In this paper, we first extend the work of [5] and [15]. We state this problem as ESMT on k-Concentric Parallel Regular n-gons.

▶ **Definition 1** (k-Concentric Parallel Regular n-gons). *k-Concentric Parallel Regular n-gons are k regular n-gons that are concentric and where the corresponding sides of polygons are parallel to each other.*

Please refer to Figure 1(a) for an example of a 2-Concentric Parallel Regular 12-gon. We call k-Concentric Parallel Regular n-gons as k-CPR n-gons for short.

We consider terminal sets where the terminals are placed on the vertices of 2-CPR n-gons. In the case of $k = 2$, the n-gon with the smaller side length will be called the inner n-gon and the other n-gon will be called the outer n-gon. Also, let a be the side length of the inner n-gon, and b be the side length of the outer n-gon. We define $\lambda = \frac{b}{a}$ and refer to it as the *aspect ratio* of the two regular polygons. In Section 3, we derive the exact structures of the SMTs for 2-CPR n-gons when the aspect ratio λ of the two polygons is greater than $\frac{1}{1-4\sin(\pi/n)}$ and $n \geq 13$.

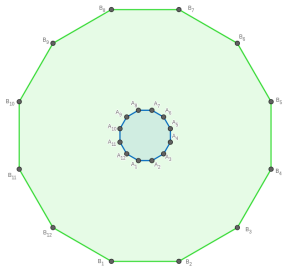
Next, we consider ESMT on an $f(n)$-Almost Convex Point Set.

▶ **Definition 2** ($f(n)$-Almost Convex Point Set). *An $f(n)$-Almost Convex Point Set $\mathcal{P}$ is a set of n points in the Euclidean plane such that there is a partition $\mathcal{P} = \mathcal{P}_1 \uplus \mathcal{P}_2$ where $\mathcal{P}_1$ forms the convex hull of $\mathcal{P}$ and $|\mathcal{P}_2| = f(n)$.*

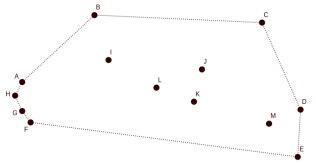
Please refer to Figure 1(b) for an example of a 5-Almost Convex Set of 13 points. In Section 4, we give an exact algorithm for ESMT on $f(n)$-Almost Convex Sets of n terminals. The running time of this algorithm is $2^{\mathcal{O}(f(n)\log n)}$. Thus, when $f(n) = \mathcal{O}(\frac{n}{\log n})$, then our algorithm runs in $2^{\mathcal{O}(n)}$ time, and when $f(n) = o(n)$ then the running time is $2^{o(n \log n)}$. This is an improvement on the best known algorithm for the general case [9].

Next in Section 5, for $f(n) = \mathcal{O}(\log n)$, we give an FPTAS. On the other hand we show that, for all $\epsilon \in (0, 1]$, when $f(n) \in \Omega(n^{\epsilon})$, there cannot exist any FPTAS unless P = NP.

Due to paucity of space, we omit certain proofs that can be found in the full version of the paper.



(a) 2-CPR 12-gons.



(b) $f(n)$-Almost Convex Point Set for $n = 13$, $f(n) = 5$.

Figure 1 Examples for Definition 1 and Definition 2.

2 Preliminaries

Notations. For a given positive integer $k \in \mathbb{N}$, the set of integers $\{1, 2, \ldots, k\}$ is denoted for short as $[k]$. Given a graph G, the vertex set is denoted as $V(G)$ and the edge set as $E(G)$. Given two graphs G_1 and G_2, $G_1 \cup G_2$ denotes the graph G where $V(G) = V(G_1) \cup V(G_2)$ and $E(G) = E(G_1) \cup E(G_2)$.

In this paper, a regular n-gon is denoted by $A_1A_2A_3...A_n$ or $B_1B_2B_3...B_n$. For convenience, we define $A_{n+1} := A_1$, $B_{n+1} := B_1$, $A_0 := A_n$ and $B_0 := B_n$. We use the notation $\{A_i\}$ to denote the polygon $A_1A_2A_3 \dots A_n$ and $\{B_i\}$ to denote the polygon $B_1B_2B_3 \dots B_n$. For any regular polygon $A_1A_2A_3...A_n$, the circumcircle of the polygon is denoted as $(A_1A_2A_3...A_n)$.

Given two points P, Q in the Euclidean plane, we denote by $\mathsf{dist}(P, Q)$ the Euclidean distance between P and Q. Given a line segment AB in the Euclidean plane, $\overline{AB} = \mathsf{dist}(A, B)$. For two distinct points A and B, L_{AB} denotes the line containing A and B; and $\overrightarrow{AB}$ denotes the ray originating from A and containing B.

When we refer to a graph $\mathcal{G}$ in the Euclidean plane then $V(\mathcal{G})$ is a set of points in the Euclidean plane, and $E(\mathcal{G})$ is a subset of the family of line segments $\{P_1P_2 | P_1, P_2 \in V(\mathcal{G})\}$. For any tree $\mathcal{T}$ in the Euclidean plane, we denote by the notation $|\mathcal{T}|$ the value of $\Sigma_{e \in E(\mathcal{T})}\overline{e}$. A path in a tree $\mathcal{T}$ is uniquely specified by the sequence of vertices on the path; therefore, P_1, P_2, P_3, …, P_k (where $P_i \in V(\mathcal{T}), \forall i \in [k]$ and $P_iP_{i+1} \in E(\mathcal{T}), \forall i \in [k-1]$) denotes the path starting from the vertex P_1, going through the vertices P_2, P_3, …, P_{k-1} and finally ending at P_k. Equivalently, we can specify the same path as *the path from P_1 to P_k*, since $\mathcal{T}$ is a tree. Consider the graph T such that $V(T) = \{v_P | P \in V(\mathcal{T})\}$, $E(T) = \{v_{P_1}v_{P_2} | P_1P_2$ is a line segment in $E(\mathcal{T})\}$. Then T is said to be the topology of $\mathcal{T}$ while $\mathcal{T}$ is said to realize the topology T. Given two trees $\mathcal{T}_1$, $\mathcal{T}_2$ in the Euclidean plane, $\mathcal{T}' = \mathcal{T}_1 \cup \mathcal{T}_2$ is the graph where $V(\mathcal{T}') = V(\mathcal{T}_1) \cup V(\mathcal{T}_2)$ and $E(\mathcal{T}') = E(\mathcal{T}_1) \cup E(\mathcal{T}_2)$.

Given any graph G, a Steiner minimal tree or SMT for a terminal set $\mathcal{P} \subseteq V(G)$ is the minimum length connected subgraph G' of G such that $\mathcal{P} \subseteq V(G')$. The Steiner Minimal Tree problem on graphs takes as input a set $\mathcal{P}$ of terminals and aims to find a minimum length SMT for $\mathcal{P}$. For the rest of the paper, we also refer to a Euclidean Steiner minimal tree as an SMT. Given a set of points $\mathcal{P}$ in the Euclidean plane, the convex hull of $\mathcal{P}$ is denoted as $\mathrm{CH}(\mathcal{P})$.

Properties of a Euclidean Steiner minimal tree. A Euclidean Steiner minimal tree (SMT) has certain structural properties as given in [3]. We state them in the following Proposition.

▶ **Proposition 3.** *Consider an SMT on n terminals.*

1. *No two edges of the SMT intersect with each other.*
2. *Each Steiner point has degree exactly 3 and the incident edges meet at 120° angles. The terminals have degree at most 3 and the incident edges form angles that are at least 120°.*
3. *The number of Steiner points is at most $n - 2$, where n is the number of terminals.*

A full Steiner tree (FST) is a Steiner tree (need not be minimal, but may include Steiner points) having exactly $n - 2$ Steiner points, where n is the number of terminals. In an FST, all terminals are leaves and Steiner points are interior nodes. When the length of an FST is minimized, it is called a minimum FST.

All SMTs can be decomposed into FST components such that, in each component a terminal is always a leaf. This decomposition is unique for a given SMT [9]. A topology for an FST is called a full Steiner topology and that of a Steiner tree is called a Steiner topology.

Steiner Hulls. A Steiner hull for a given set of points is defined to be a region which is known to contain an SMT. We get the following propositions from [9].

▶ **Proposition 4.** *For a given set of terminals, every SMT is always contained inside the convex hull of those points. Thus, the convex hull is also a Steiner hull.*

The next two propositions are useful in restricting the structure of SMTs and the location of Steiner points.

▶ **Proposition 5** (The Lune property). *Let* UV *be any edge of an SMT. Let* $L(\mathrm{U},\mathrm{V})$ *be the lune-shaped intersection of circles of radius* $|\mathrm{UV}|$ *centered on* U *and* V*. No other vertex of the SMT can lie in* $L(\mathrm{U},\mathrm{V})$*, except* U *and* V *themselves.*

▶ **Proposition 6** (The Wedge property). *Let* W *be any open wedge-shaped region having angle* $120°$ *or more and containing none of the points from the input terminal set* $\mathcal{P}$*. Then* W *contains no Steiner points from an SMT of* $\mathcal{P}$*.*

Fully Polynomial Time Approximation Scheme (FPTAS). An algorithm is called a fully polynomial time approximation scheme (FPTAS) for a problem if it takes an input instance and a parameter $\epsilon > 0$, and outputs a solution with approximation factor $(1+\epsilon)$ for a minimization problem in time $(1/\epsilon)^{\mathcal{O}(1)} n^{\mathcal{O}(1)}$ where n is the input size.

3 Polynomial cases for Euclidean Steiner Minimal Tree

In this section, we consider the Euclidean Steiner Minimal Tree problem for 2-CPR n-gons. Throughout the section, we denote the inner n-gon as $\{A_i\}$ and the outer n-gon as $\{B_i\}$. First, we consider the configuration of an Euclidean Steiner minimal tree in a subsection of the annular area between $\{A_i\}$ and $\{B_i\}$, which will form an isosceles trapezoid.

Then we prove our result for all 2-CPR n-gons.

3.1 Isosceles Trapezoids and Vertical Forks

In this section, we discuss one particular Steiner topology when the terminal set is formed by the four corners of a given isosceles trapezoid. However, we will limit the discussion to only the isosceles trapezoids such that the angle between the non-parallel sides is of the form $\frac{2\pi}{n}$ where $n \in \mathbb{N}$, $n \geq 4$. The reason is that given 2-CPR n-gons $\{A_i\}$, $\{B_i\}$, for $n \geq 4$ and for any $j \in \{1,\ldots,n-1\}$, the region $A_jA_{j+1}B_jB_{j+1}$ is an isosceles trapezoid such that the angle between the non-parallel sides is of the form $\frac{2\pi}{n}$.

(a) Isosceles trapezoid with $\angle AOB = \frac{2\pi}{8}$. **(b)** The Vertical Fork, $\mathcal{T}_{vf}$.

Figure 2 Isosceles Trapezoids and Vertical Forks.

Let $ABQP$ be an isosceles trapezoid with AB, PQ as the parallel sides, and AP, BQ as the non-parallel sides. Assume without loss of generality that AB is shorter than PQ. Let $\overline{AB} = 1$ and $\overline{PQ} = \lambda$, where $\lambda \geq \frac{\sqrt{3}+\tan\frac{\pi}{n}}{\sqrt{3}-\tan\frac{\pi}{n}}$. For brevity, we say $\lambda_v = \frac{\sqrt{3}+\tan\frac{\pi}{n}}{\sqrt{3}-\tan\frac{\pi}{n}}$. Let L_{PA} and L_{QB} be the lines containing the line segments PA and QB respectively. Also let O be the point of intersection of L_{PA} and L_{QB}. Further, let M and N be the midpoints of AB and PQ respectively (as in Figure 2(a)). As mentioned earlier, $\angle AOB = \frac{2\pi}{n}$ where $n \in \mathbb{N}$, $n \geq 4$.

Now, we explore the following Steiner topology of the terminal set $\{A, B, P, Q\}$:

1. A and B are connected to a Steiner point S_1.
2. P and Q are connected to another Steiner point S_2.
3. S_1 and S_2 are directly connected (Please see Figure 2(b)).

We call such a topology a *vertical fork topology* and the Steiner tree realising such a topology, the *vertical fork*. Note that in a vertical fork topology the only unknowns are the locations of the two Steiner points S_1, S_2. Therefore, we have the vertical fork topology as T_{vf}, with $E(T_{vf}) = \{AS_1, BS_1, S_1S_2, S_2P, S_2Q\}$. We show the existence of a vertical fork and calculate its total length in the following lemma.

▶ **Lemma 7.** *A vertical fork $\mathcal{T}_{vf}$ can be constructed for any $n \geq 4$ and for any $\lambda \geq \lambda_v$, where*

$$\lambda_v = \frac{\sqrt{3} + \tan\frac{\pi}{n}}{\sqrt{3} - \tan\frac{\pi}{n}}$$

such that the length of the vertical fork

$$|\mathcal{T}_{vf}| = \frac{(\lambda - 1)}{2\tan\frac{\pi}{n}} + \frac{\sqrt{3}(\lambda + 1)}{2}$$

3.2 Euclidean Steiner Minimal Tree and Large Polygons with Large Aspect Ratios

In this section, we consider the Euclidean Steiner Minimal Tree problem when the terminal set is formed by the vertices of 2-CPR n-gons, namely $\{A_i\}$ and $\{B_i\}$. As mentioned earlier, $\{A_i\}$ is the inner polygon and $\{B_i\}$ is the outer polygon of this set of 2-CPR n-gons. In particular, we consider the case when $n \geq 13$; for $n \leq 12$ these are constant sized input instances and can be solved using any brute-force technique. We also require that the aspect ratio λ has a lower bound λ_1, i.e. we do not want the two polygons to have sides of very similar length. The exact value of λ_1 will be clear during the description of the algorithm. Intuitively, when λ is *very large*, the SMT should look similar to what was derived in [15]. We call the topology of such an SMT a *singly connected topology*, as in Figure 3.

▶ **Definition 8.** *A Steiner topology of $\{A_i\} \cup \{B_i\}$ is a **singly connected topology**, if it has the following structure:*

- *A vertical gadget i.e. five edges $\{A_jS_a, A_{j+1}S_a, S_aS_b, S_bB_j, S_bB_{j+1}\}$ for some $1 \leq j \leq n$, where S_a and S_b are newly introduced Steiner points contained in the isosceles trapezoid $\{A_j, A_{j+1}, B_j, B_{j+1}\}$.*
- *All $(n-2)$ polygon edges of $\{A_i\}$ excluding the edge A_jA_{j+1}*
- *All $(n-2)$ polygon edges of $\{B_i\}$ excluding the edge B_jB_{j+1}*

For the rest of this section, we consider the SMTs for a large enough aspect ratio, λ and show that there is an SMT that must be a realisation of a *singly connected topology*. We refer to an SMT for the terminal set defined by the vertices of $\{A_i\}$ and $\{B_i\}$ as the SMT for $\{A_i\} \cup \{B_i\}$.

Without loss of generality, we consider the edge length of any side A_iA_{i+1} in $\{A_i\}$ to be 1 and that of any side B_iB_{i+1} of $\{B_i\}$ to be λ.

We define the notion of a path in an SMT for the vertices of $\{A_i\}$ and $\{B_i\}$ where the starting point is in $\{A_i\}$ and the ending point is in $\{B_i\}$.

▶ **Definition 9.** *An **A-B path** is a path in a Steiner tree of $\{A_i\} \cup \{B_i\}$ which starts from a vertex in $\{A_i\}$ and ends at a vertex in $\{B_i\}$ with all intermediate nodes (if any) being Steiner points.*

(a) SMT of 2-CPR 13-gons. **(b)** SMT of 2-CPR 20-gons.

Figure 3 *Singly connected topology* of 2-CPR n-gons $n = 13$ and $n = 20$.

The following Definition and Figure 4 is useful for the design of our algorithm.

▶ **Definition 10.** *A* ***counter-clockwise path*** *is a path $P_1, P_2, ...P_m$ in a Steiner tree such that for all $i \in \{2, \ldots, m-1\}, \angle P_{i-1}P_iP_{i+1} = \frac{2\pi}{3}$ in the counter-clockwise direction. Similarly, a* ***clockwise path*** *is a path $P_1, P_2, ...P_m$ in a Steiner tree such that for all $i \in \{2, \ldots, m-1\}, \angle P_{i-1}P_iP_{i+1} = \frac{4\pi}{3}$ in the counter-clockwise direction.*

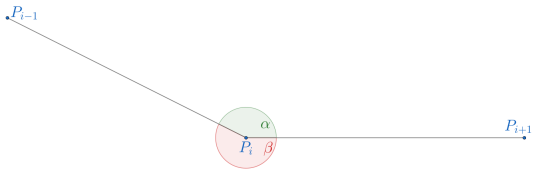


Figure 4 $\angle P_{i-1}P_iP_{i+1} = \alpha$ in the **counter-clockwise direction** and $\angle P_{i-1}P_iP_{i+1} = \beta$ in the **z**, as in Definition 10.

Now, we consider any Steiner point S in any SMT. Let P and Q be two neighbours of S. We now prove that there is no point of the SMT inside the triangle PSQ.

▶ **Observation 11.** *Let S be a Steiner point in any SMT for $\{A_i\} \cup \{B_i\}$, with neighbours P and Q. Then, no point of the SMT lies inside the triangle PSQ.*

Next, we show that in an SMT for $\{A_i\} \cup \{B_i\}$ there cannot be any Steiner point, in the interior of the polygon $\{A_i\}$, that is a direct neighbour of some point B_k in the polygon $\{B_i\}$.

▶ **Observation 12.** *For any SMT for $\{A_i\} \cup \{B_i\}$, there cannot exist a Steiner point S lying in the interior of the polygon $\{A_i\}$ such that SB_k is an edge in an SMT for some $B_k \in \{B_i\}$.*

Proof. For the sake of contradiction, we assume that for some SMT for $\{A_i\} \cup \{B_i\}$ there exists a Steiner point S lying in the interior of the polygon $\{A_i\}$ such that SB_k is an edge in the SMT for some $B_k \in \{B_i\}$. Let A_mA_{m+1} be the edge such that SB_k intersects A_mA_{m+1}. Without loss of generality, assume that A_m is closer to B_k than A_{m+1}. Therefore $\angle B_kA_mS > \angle B_kA_mA_{m+1} \geq \frac{\pi}{2} + \frac{\pi}{n} > \frac{\pi}{2}$. This means that B_kS is the longest edge in the triangle B_kSA_m. Therefore we can remove the edge B_kS from the SMT and replace it with either B_kA_m or SA_m to get another tree connecting the terminal set with a shorter total length than what we started with, which is a contradiction. ◀

We further analyze SMTs for $\{A_i\} \cup \{B_i\}$.

▶ **Observation 13.** *Let $\mathcal{V} = \{A_j, A_{j+1}, \ldots, A_k\}$ be the interval of consecutive vertices of $\{A_i\}$ lying between A_j and A_k (which includes A_{j+1}) such that A_j is distinct from A_{k+1}. Let U be any point on the line segment A_kA_{k+1}. Then an SMT of $\mathcal{V} \cup \{U\}$ is $\mathcal{T}$, with $E(\mathcal{T}) = \{A_jA_{j+1}, A_{j+1}A_{j+2}, \ldots, A_{k-1}A_k\} \cup \{A_kU\}$.*

We proceed by showing that in any SMT for $\{A_i\} \cup \{B_i\}$ there exists at least one A-B path which is also a counter-clockwise path. Symmetrically, we also show that for any SMT for $\{A_i\} \cup \{B_i\}$ there exists another clockwise A-B path which consists of only clockwise turns. We can intuitively see that this is true because, if all clockwise paths starting at a vertex in $\{A_i\}$ also ended in a vertex in $\{A_i\}$, there would be enough paths to form a cycle, which is not possible in a tree.

▶ **Lemma 14.** *In any SMT for $\{A_i\} \cup \{B_i\}$, there exists an A-B path which is also a clockwise path and there exists an A-B path which is also a counter-clockwise path.*

Our next step is to bound the number of "connections" that connect the inner polygon $\{A_i\}$ and the outer polygon $\{B_i\}$ for a large aspect ratio, λ. As λ increases, the area of the annular region between the two polygons increases as well. Therefore, an increase in the number of connections would lead to a longer total length of the SMT considered. Consequently, we will prove that after a certain positive constant λ_1, for $\lambda > \lambda_1$ any SMT for $\{A_i\} \cup \{B_i\}$ will have a single "connection" between the two polygons. Moreover, [15] gives us an evidence that as $\lambda \to \infty$, there will indeed be a single connection connecting the outer polygon and the inner polygon for $n \geq 12$. We can formalize this notion of existence of a single "connection" with the following lemma.

▶ **Lemma 15.** *For any SMT for $\{A_i\} \cup \{B_i\}$ with $n \geq 13$ and $\lambda > \lambda_1$, the number of edges needed to be removed in order to disconnect $\{A_i\}$ and $\{B_i\}$ is 1, where*

$$\lambda_1 = \frac{1}{1 - 4\sin\frac{\pi}{n}}$$

We now proceed to further investigate the connectivity of $\{A_i\}$ and $\{B_i\}$.

▶ **Lemma 16.** *Consider an SMT for $\{A_i\} \cup \{B_i\}$ for $n \geq 13$ and $\lambda \geq \lambda_1$. There must exist $j \in [n]$ and a Steiner point S_1, such that terminals A_j, A_{j+1} form a path A_j, S_1, A_{j+1} in the SMT and each A-B path passes through S_1.*

Proof Sketch. From Lemma 14, we know that there exists one clockwise A-B path and one counterclockwise A-B path in any SMT of $\{A_i\} \cup \{B_i\}$. Let a clockwise A-B path start from A_r and a counter-clockwise A-B path start from A_l. Further following from Lemma 15, as there is one edge common to all A-B paths, the clockwise A-B path from A_r and the counter-clockwise A-B path from A_l must share a common edge S_1S_2. Therefore, each A-B path must pass through S_1 and S_2. Without loss of generality we assume that point S_1 is closer to the polygon $\{A_i\}$ than S_2. This means S_1 is either a Stiener point or a terminal vertex of $\{A_i\}$.

▷ Claim 17. S_1 is not a vertex in $\{A_i\}$.

Therefore, S_1 must be a Steiner point. Let P and Q be the neighbours of S_1 other than S_2, such that $\angle PS_1S_2$ is a clockwise turn while $\angle QS_1S_2$ is a counter-clockwise turn. This means that the clockwise A-B path from A_r passes through P and the counter-clockwise A-B path from A_l passes through Q. We prove the following for P and Q.

▷ Claim 18. P and Q are consecutive vertices of $\{A_i\}$ in any SMT of $\{A_i\} \cup \{B_i\}$.

Therefore, P and Q are consecutive vertices A_j, A_{j+1} of the polygon $\{A_i\}$, for some $j \in [n]$ such that A_j, S_1, A_{j+1} is a path in the SMT, where S_1 is a Steiner point lying on all A-B paths. ◀

Our next step is to investigate some more structural properties of an SMT for $\{A_i\} \cup \{B_i\}$. From [5], we may guess that there would be a lot of polygon edges of both $\{A_i\}$ and $\{B_i\}$ in an SMT. We prove the following Lemma, stating that there is an SMT of $\{A_i\} \cup \{B_i\}$ which contains $(n-2)$ polygon edges of $\{A_i\}$.

▶ **Lemma 19.** *For an SMT for $\{A_i\} \cup \{B_i\}$ with aspect ratio λ, $\lambda > \lambda_1 = \frac{1}{1-4\sin\frac{\pi}{n}}$, let S_1 be the Steiner point such that all A-B paths pass through S_1. Let A_j and A_{j+1} be vertices of $\{A_i\}$ which are connected to S_1. Then, there exists an SMT of $\{A_i\} \cup \{B_i\}$ having $n-2$ polygon edges of $\{A_i\}$ other than A_jA_{j+1}.*

With these set of results in hand, we can now show that there exists an SMT of $\{A_i\} \cup \{B_i\}$ following a *singly connected topology*. To show this, we start with any SMT for $\{A_i\} \cup \{B_i\}$, $\mathcal{T}_0$, that satisfies all the results derived so far and transform it into a Steiner tree of *singly connected topology* having total length not longer than the initial Steiner tree $\mathcal{T}_0$.

▶ **Theorem 20.** *There exists an SMT for $\{A_i\} \cup \{B_i\}$ following a singly connected topology for $n \geq 13$ and $\lambda \geq \lambda_1$, where*

$$\lambda_1 = \frac{1}{1 - 4\sin\frac{\pi}{n}}$$

Proof. Let $\mathcal{T}_0$ be any SMT of $\{A_i\} \cup \{B_i\}$ which satisfies the properties of Lemma 19. Further, from Lemma 16, there is a Steiner point S_1 which lies on all A-B paths, and there are two consecutive vertices A_j, A_{j+1} such that A_j, S_1, A_{j+1} is a path in $\mathcal{T}_0$. As $\mathcal{T}_0$ satisfies the property of Lemma 19, $\mathcal{T}_0$ has $n-2$ polygon edges of $\{A_i\}$ excluding the edge A_jA_{j+1}.

Let H be the point in the interior of the polygon $\{A_i\}$ such that HA_jA_{j+1} form an equilateral triangle. As $n > 6$, the common centre O of $\{A_i\}$ and $\{B_i\}$ does not lie inside the triangle HA_jA_{j+1}. Now, we modify $\mathcal{T}_0$ as follows:

1. Remove edges A_jS_1, S_1A_{j+1} and add edge S_1H to get the forest $\mathcal{T}_1$. We know from [9] that S_1, S_2 and H are collinear and this transformation does not change the total length. Therefore $|\mathcal{T}_0| = |\mathcal{T}_1|$. Here, $|\mathcal{T}_1|$ denotes the sum of the lengths of edges present in $\mathcal{T}_1$.
2. Add edge HO and remove all polygon edges of $\{A_i\}$ to get $\mathcal{T}_2$. Therefore $|\mathcal{T}_2| = |\mathcal{T}_1| + \overline{HO} - (n-2) = |\mathcal{T}_0| + \overline{HO} - (n-2)$. We observe that $\mathcal{T}_2$ is a tree connecting the points in $\{B_i\} \cup \{O\}$.
3. Let S_0 be the Torricelli point of the triangle OB_jB_{j+1}. Let $\mathcal{T}_3$ be the Steiner tree of $\{B_i\} \cup \{O\}$ with edges S_0O, S_0B_j, S_0B_{j+1} and other points in $\{B_i\}$ connected through $(n-2)$ polygon edges of the polygon $\{B_i\}$. From [15], we know that $\mathcal{T}_3$ is the SMT of $\{B_i\} \cup \{O\}$. Therefore $|\mathcal{T}_3| \leq |\mathcal{T}_2| = |\mathcal{T}_0| + \overline{HO} - (n-2)$. Further we know that H lies on the edge OS_0 (as O, S_0 and H lie on the perpendicular bisector of B_j and B_{j+1}).
4. Remove edge S_0O and add edge S_0H to get $\mathcal{T}_4$. As H lies on the edge OS_0, we have $|\mathcal{T}_4| = |\mathcal{T}_3| - \overline{OH} \leq |\mathcal{T}_0| - (n-2)$.
5. Let S_3 be the intersection of the circumcircle of triangle A_jHA_{j+1} (from Lemma 7 the intersection exists as $\lambda_1 \geq \lambda_v$ for $n \geq 13$). Remove the edge S_3H and add the edges S_3A_j and S_3A_{j+1} to get $\mathcal{T}_5$. Again, from [9] we know that this transformation does not change the total length. Hence $|\mathcal{T}_5| = |\mathcal{T}_5| \leq |\mathcal{T}_0| - (n-2)$. Moreover, as $\lambda > \lambda_v$, we observe that $\{A_j, B_j, A_{j+1}, B_{j+1}, S_3, S_0\}$ form the vertices of the vertical gadget and points O, H, S_3, S_0 appear in that order on the perpendicular bisector of B_j and B_{j+1}.

6. Add back the $(n-2)$ polygon edges of $\{A_i\}$ which were removed in the second step to get $\mathcal{T}_6$. Therefore $|\mathcal{T}_4| = |\mathcal{T}_5| + (n-2) \leq |\mathcal{T}_0|$. We further observe that $\mathcal{T}_6$ is a Steiner tree connecting the points $\{A_i\} \cup \{B_i\}$ with a singly connected topology.

Therefore we started with an arbitrary SMT $\mathcal{T}_0$ and transformed it into a Steiner tree $\mathcal{T}_6$ with a singly connected topology (where $\{A_j, B_j, A_{j+1}, B_{j+1}, S_3, S_0\}$ form the vertices of the vertical gadget) which has a total length not worse than $\mathcal{T}_0$. Hence $\mathcal{T}_6$ must be an SMT of $\{A_i\} \cup \{B_i\}$. This proves the theorem. ◀

▶ Remark 21. Theorem 20 determines the exact structure of the SMT for $\{A_i\} \cup \{B_i\}$. Further from Section 3.1 we determine the exact method to construct the two additional Steiner points in $\mathcal{O}(1)$ steps - note that this construction time is independent of the integer n or the real number λ. Therefore, SMT for $\{A_i\} \cup \{B_i\}$ for $n \geq 13$ and $\lambda \geq \lambda_1$ is solvable in polynomial time.

Interestingly, λ_1 converges to 1 very quickly with increasing n:

n	13	20	40	100	500
λ_1	23.3987	2.6719	1.4574	1.1437	1.0258

4 Euclidean Steiner Minimal Tree on $f(n)$-Almost Convex Point Sets

In this section, we design an exact algorithm for Euclidean Steiner Minimal Tree on $f(n)$-Almost Convex Point Sets running in time $2^{\mathcal{O}(f(n)\log n)}$. Note that $f(n) \leq n$ is always true. Therefore, we are given as input a set $\mathcal{P}$ of n points in the Euclidean Plane such that $\mathcal{P}$ can be partitioned as $\mathcal{P} = \mathcal{P}_1 \uplus \mathcal{P}_2$, where $\mathcal{P}_1$ is the convex hull of $\mathcal{P}$ and $|\mathcal{P}_2| = f(n)$.

We know that the SMT of $\mathcal{P}$ can be decomposed uniquely into one or more full Steiner subtrees, such that two full Steiner subtrees share at most one node [9]. In the following lemma, we further characterize one full Steiner subtree, which we refer to as a *leaf full Steiner subtree*.

▶ **Lemma 22.** *Let $\mathbb{F}$ be the full Steiner decomposition of an SMT of $\mathcal{P}$. Then there exists a full Steiner subtree $\mathcal{F} \in \mathbb{F}$ such that $\mathcal{F}$ has at most one common node with at most one other full Steiner subtree in $\mathbb{F}$.*

Using the above Lemma, along with the bounds on the number of FSTs from [9], we can obtain the following theorem (details can be found in the full version).

▶ **Theorem 23.** *An SMT $\mathcal{T}_\mathcal{P}$ of a k-Almost Convex Set $\mathcal{P}$ of terminals can be computed in $\mathcal{O}(n^k \cdot 5^n)$ time.*

The above theorem gives us several improvements in special classes of inputs.

▶ **Corollary 24.** *Let $\mathcal{P}$ be a $f(n)$-Almost Convex Point Set. Then, then there is an algorithm $\mathcal{A}$ for Euclidean Steiner Minimal Tree such that, $\mathcal{A}$ runs in $2^{\mathcal{O}(n+f(n)\log n)}$ time. In particular,*

1. *When $f(n) = \mathcal{O}(n)$, $\mathcal{A}$ runs in $2^{\mathcal{O}(n\log n)}$ time.*
2. *When $f(n) = \Omega(\frac{n}{\log n})$ and $f(n) = o(n)$, $\mathcal{A}$ runs in $2^{o(n\log n)}$.*
3. *When $f(n) = \mathcal{O}(\frac{n}{\log n})$, $\mathcal{A}$ runs in $2^{\mathcal{O}(n)}$ time.*

Therefore, for $f(n) = o(n)$, our algorithm for Euclidean Steiner Minimal Tree does better on $f(n)$-Almost Convex Points Sets than the current best known algorithm [8].

5 Approximation Algorithms for Euclidean Steiner Minimal Tree

The Euclidean Steiner Minimal Tree problem is NP-hard as shown by Garey et al. in [6]. Garey et al. also prove that there cannot be an FPTAS (fully polynomial time approximation scheme) for this problem unless $P = NP$. At the same time, the case when all the terminals lie on the boundary of a convex region admits an FPTAS as given in [14]. In this section, we aim to conduct a more fine-grained analysis for the problem by considering $f(n)$-Almost Convex Point Sets of n terminals and studying the existence of FPTASes for different functions $f(n)$.

First, we present an FPTAS for Euclidean Steiner Minimal Tree on $f(n)$-Almost Convex Sets of n terminals, when $f(n) = \mathcal{O}(\log n)$. The FPTAS follows the strategy of [14] and uses a variant of the Dreyfus-Wagnus Steiner tree algorithm [4] as a subroutine.

▶ **Theorem 25.** *There exists an FPTAS for Euclidean Steiner Minimal Tree on an $f(n)$-Almost Convex Set of n terminals, where $f(n) = \mathcal{O}(\log n)$.*

On the other hand, we prove in the next section that no FPTAS exists for the case when $f(n) = \Omega(n^\epsilon)$, where $\epsilon \in (0, 1]$.

5.1 Hardness of Approximation for Euclidean Steiner Minimal Tree on Cases of Almost Convex Sets

In this section, we consider the Euclidean Steiner Minimal Tree problem on $f(n)$-Almost Convex Sets of n terminal points, where $f(n) = \Omega(n^\epsilon)$ for some $\epsilon \in (0, 1]$. We show that this problem cannot have an FPTAS. The proof strategy is similar to that in [6]. First, we give a reduction for the problem Exact Cover by 3-Sets (defined below) to our problem to show that our problem is NP-hard. Next, we consider a discrete version of our problem and reduce our problem to the discrete version. The discrete version is in NP. Therefore, this chain of reductions imply that the discrete version of our problem is Strongly NP-complete and therefore cannot have an FPTAS, following from [6]. Similar to the arguments in [6], this also implies that our problem cannot have an FPTAS.

Before we describe our reductions, we take a look at the NP-hardness reduction of the Euclidean Steiner Minimal Tree problem from the Exact Cover by 3-Sets (X3C) problem in [6]. In the X3C problem, we are given a universe of elements $U = \{1, 2, \ldots, 3n\}$ and a family $\mathbb{F}$ of 3-element subsets $F_1, F_2, \ldots, F_t$ of the $3n$ elements. The objective is to decide if there exists a subcollection $\mathbb{F}' \subseteq \mathbb{F}$ such that: (i) the elements of $\mathbb{F}'$ are disjoint, and (ii) $\bigcup_{F' \in \mathbb{F}'} F' = U$. The X3C problem is NP-complete [7].

In [6], various gadgets are constructed, i.e. particular arrangements of a set of points. These are then arranged on the plane in a way corresponding to the given X3C instance. Figure 5 shows the reduced ESMT instance obtained for $U = \{1, 2, 3, 4, 5, 6\}$ and $\mathbb{F} = \{\{1, 2, 4\}, \{2, 3, 6\}, \{3, 5, 6\}\}$ (taken from [6]). The squares, hexagons (crossovers), shaded circles (terminators) and lines (rows) all represent specific arrangements of a subset of points. Let $X(\mathbb{F})$ denote the reduced instance. The number of points in $X(\mathbb{F})$ is bounded by a polynomial in n and t. Let this polynomial be $\mathcal{O}(t^\gamma)$, as we can assume $t \geq n$ since otherwise it trivially becomes a NO instance. Here γ is some constant.

We restate Theorem 1 in [6].

▶ **Proposition 26.** *Let $\mathcal{S}^*$ denote an SMT of $X(\mathbb{F})$, the instance obtained by reducing the X3C instance $(n, \mathbb{F})$, and $|\mathcal{S}^*|$ denote its length. If $\mathbb{F}$ has an exact cover, then $|\mathcal{S}^*| \leq f(n, t, \hat{C})$, otherwise $|\mathcal{S}^*| \geq f(n, t, \hat{C}) + \frac{1}{200nt}$, where $t = |\mathbb{F}|$, $\hat{C}$ is the number of crossovers, i.e. hexagonal gadgets, and f is a positive real-valued function of $n, t, \hat{C}$.*

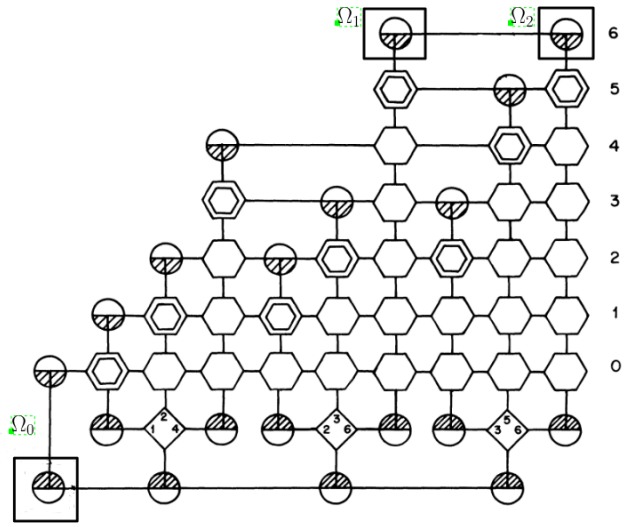


Figure 5 Reduced instance of ESMT from X3C (taken from [6]).

We extend this construction to prove NP-hardness for instances of Euclidean Steiner Minimal Tree where the terminal set $\mathcal{P}$ has $\Omega(n^\epsilon)$ points inside $\text{CH}(\mathcal{P})$. Here, $\epsilon \in (0, 1]$ and n is the number of terminals.

Let us call the *length* of a gadget to be the maximum horizontal distance between any two points in that gadget. Similarly, we define the *breadth* of a gadget to be the maximum vertical distance between any two points in that gadget.

(a) The Upward Terminator symbol.

(b) The Downward Terminator symbol.

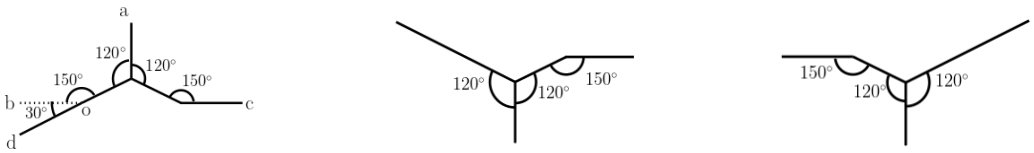
(c) The Downward Terminator symbol.

Figure 6 The Terminator gadget symbol and arrangement of points.

The *terminator* gadget used is shown in Figure 6. The straight lines represent a row of at least 1000 points separated at distances of 1/10 or 1/11. The angles between them are as shown. The upward terminator has the point A above the other points in the terminator, whereas the downward terminator has the point A below the other points. Firstly, we adjust the number of points in the long rows, such that the length and breadth of the terminators is same as that of the hexagonal gadgets (crossovers). We can fix this length and breadth to be some constants, such that the number of points in each gadget is also bounded by some constant. In our construction, we modify the terminators Ω_0, Ω_1, and Ω_2 as shown in Figure 5 enclosed in squares. Ω_1 is the terminator corresponding to the first occurrence of the element $3n \in U$ in some set in $\mathbb{F}$ and Ω_2 is the terminator corresponding to the last occurrence of $3n$ in some set in $\mathbb{F}$ (if there are more than one occurrences of $3n$). If there are no occurrences of $3n$, then it is trivially a no-instance. The modified gadgets are shown in Figure 7. All the other gadgets remain unaltered.

We define a Conic Set of points.

▶ **Definition 27.** *A Conic Set is a set of points consisting of a point T, called the tip of the cone, and the remaining points denoted by $\mathcal{S}$. Let $\mathcal{C}$ be the circle with T as centre and radius r. All the points in $\mathcal{S}$ lie on $\mathcal{C}$, such that the angle at the tip formed by the two extreme points $L, R \in \mathcal{S}$, i.e. $\angle LTR = 30°$ in the anticlockwise direction. So, we have $\overline{TL} = \overline{TR} = r$. The distance between any two consecutive points in $\mathcal{S}$ is the same, say d. Let the number of points in $\mathcal{S}$ be n. We denote the Conic Set as $\text{Cone}(T, r, n)$ and $\mathcal{S}$ as $\text{Circ}(T, r, n)$. We call TL as the left slope of the Conic Set and TR as the right slope of the Conic Set.*



(a) Ω'_0. **(b)** Ω'_1. **(c)** Ω'_2.

Figure 7 The modified terminator gadgets.

We use the Conic Set in the reduction for our problem (please see Figure 8). Now, we give a sketch of the reduction of an X3C instance $(n, \mathbb{F})$ to an instance $X'(\mathbb{F}, \epsilon)$ of ESMT.

Algorithm $\mathcal{A}$ for construction of ESMT instance $X'(\mathbb{F}, \epsilon)$ from X3C instance $(n, \mathbb{F})$:

- Reduce the input X3C instance to the points configuration $X(\mathbb{F})$ according to the reduction given in [6].
- Let $DQCP$ be the smallest axis-parallel rectangle bounding $X(\mathbb{F})$ after certain modifications of gadgets described in [6] (details in the full version).
- Take $\alpha = \frac{1}{\epsilon}$. Define $r = ct^\alpha = \mathcal{O}(t^\alpha)$ and $n' = c't^{\gamma\alpha} = \mathcal{O}(t^{\gamma\alpha})$, where $t = |\mathbb{F}|$ and c and c' are constants. Add the $\text{Cone}(D, r, n')$, such that D is the tip of the Conic Set, and the left slope DE makes an angle of 120° with DP. The right slope DF also makes an angle of 120° with DQ.

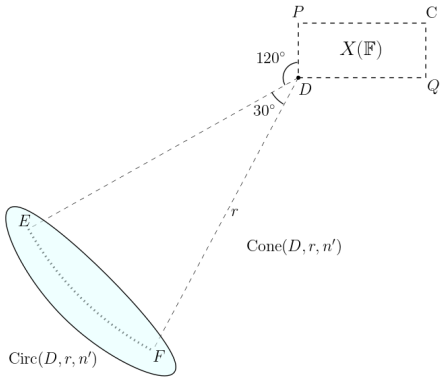


Figure 8 The reduced instance $X'(\mathbb{F}, \epsilon)$.

Now we state a few properties of the constructed instance $X'(\mathbb{F}, \epsilon)$ (detailed proofs can be found in the full version).

▶ **Lemma 28.** *All the points in* $\mathrm{Circ}(D, r, n')$ *(according to Definition 27) lie on the convex hull of the reduced ESMT instance* $X'(\mathbb{F}, \epsilon)$ *constructed by Algorithm* $\mathcal{A}$*, where* $\epsilon \in (0, 1]$.

Let us denote the convex hull of $X'(\mathbb{F}, \epsilon)$ by $\mathrm{CH}(X'(\mathbb{F}, \epsilon))$ and that of the points lying inside or on the bounding rectangle PDQC, i.e. $X'(\mathbb{F}, \epsilon) \setminus \mathrm{Circ}(D, r, n')$ by $\mathrm{CH}(X'(\mathbb{F}, \epsilon) \setminus \mathrm{Circ}(D, r, n'))$.

▶ **Lemma 29.** *The reduced ESMT instance* $X'(\mathbb{F}, \epsilon)$ *constructed by Algorithm* $\mathcal{A}$ *has* $\Omega(N^{\epsilon})$ *points inside the convex hull, where* $\epsilon \in (0, 1]$ *and* N *is the total number of terminals in* $X'(\mathbb{F}, \epsilon)$.

We further state structural properties of SMTs of the reduced instance $X'(\mathbb{F}, \epsilon)$ when considering the modified gadgets Ω'_0, Ω'_1, and Ω'_2.

▶ **Lemma 30.** *Consider an SMT* $\mathcal{S}^*$ *of the ESMT instance* $X(\mathbb{F})$ *obtained via reduction from the X3C instance* $(n, \mathbb{F})$ *as per [6]. Consider a tree* $\mathcal{S}'^*$ *on the terminal set of* $X'(\mathbb{F}, \epsilon)$ *obtained from* $\mathcal{S}^*$ *as follows: Consider the modified terminator gadgets* Ω'_i, $i \in \{0, 1, 2\}$ *as in Algorithm* $\mathcal{A}$*. For each* $i \in \{0, 1, 2\}$*, the edge* B_iO_i *is excluded from* $\mathcal{S}^*$ *and the edge* D_iO_i *is included to form* $\mathcal{S}'^*$*.* $\mathcal{S}'^*$ *is an SMT for the terminal set of* $X'(\mathbb{F}, \epsilon)$.

Now we focus on the structure of the SMT of $X'(\mathcal{F}, \epsilon)$. The SMT is basically the union of the SMT $\mathcal{S}'^*$ of the points in the bounding rectangle $PDQC$ as stated in Lemma 30 and the SMT of the set of points $\mathrm{Cone}(D, r, n')$.

$\mathrm{CH}(X'(\mathbb{F}, \epsilon) \setminus \mathrm{Circ}(D, r, n'))$ is enclosed by the bounding rectangle $PDQC$ and D must lie on $\mathrm{CH}(X'(\mathbb{F}, \epsilon) \setminus \mathrm{Circ}(D, r, n'))$. We label the vertices of $\mathrm{CH}(X'(\mathbb{F}, \epsilon) \setminus \mathrm{Circ}(D, r, n'))$ as $D, P_1, P_2, \ldots, P_k$ in the counter-clockwise order. Let $\mathrm{CH}(X'(\mathbb{F}, \epsilon))$ be the convex hull of all the points. By Lemma 28, all the points in $\mathrm{Circ}(D, r, n')$ lie on $\mathrm{CH}(X'(\mathbb{F}, \epsilon))$. Let EP_i and FP_j be edges in $\mathrm{CH}(X'(\mathbb{F}, \epsilon))$, such that $P_i, P_j \notin \mathrm{Circ}(D, r, n')$.

The SMT of $X'(\mathbb{F}, \epsilon)$ clearly lies inside its convex hull, $\mathrm{CH}(X'(\mathbb{F}, \epsilon))$. We show that the Steiner hull can be further restricted to the bounding rectangle $PDQC$ and the convex polygon formed by the points in $\mathrm{Cone}(D, r, n')$. For this we use Theorem 1.5 in [9], as stated below.

▶ **Proposition 31** ([9])**.** *Let* H *be a Steiner hull of* N*. By sequentially removing wedges* abc *from the remaining region, where* a, b, c *are terminals but* $\triangle$abc *contains no other terminal,* a *and* c *are on the boundary and* $\angle\mathrm{abc} \geq 120°$*, a Steiner hull* H' *invariant to the sequence of removal is obtained.*

▶ **Lemma 32.** *The region comprising of the bounding rectangle* $PDQC$ *according to Algorithm* $\mathcal{A}$ *and the convex polygon formed by the set of points* $\mathrm{Cone}(D, r, n')$ *is a Steiner hull of* $X'(\mathbb{F}, \epsilon)$.

Given the nature of the above Steiner hull, we show that we can treat $X(\mathbb{F})$ and $\mathrm{Cone}(D, r, n')$ separately.

▶ **Lemma 33.** *There is an SMT of* $X'(\mathbb{F}, \epsilon)$ *that is the union of an SMT of* $X(\mathbb{F})$ *and an SMT of the points in* $\mathrm{Cone}(D, r, n')$*, with* D *being common to both of them.*

We can identify a structure for an SMT of the points in $\mathrm{Cone}(D, r, n')$ using [15].

▶ **Lemma 34.** *There is an SMT of the points in* $\mathrm{Cone}(D, r, n')$ *that is as shown in Figure 9. In the SMT,* D *is connected to the two middle points in* $\mathrm{Circ}(D, r, n')$ *via a Steiner point* S^t*. The other points in* $\mathrm{Circ}(D, r, n')$ *are connected along the circumference.*

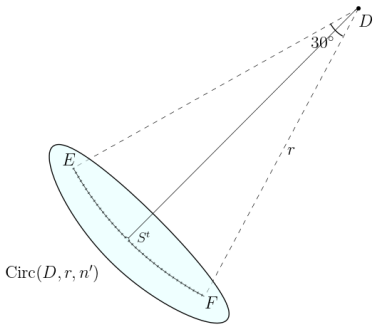


Figure 9 SMT of Cone(D, r, n').

Finally, we prove the NP-hardness of Euclidean Steiner Minimal Tree on $f(n)$-Almost Convex Sets of n terminals, when $f(n) = \Omega(n^\epsilon)$ for some $\epsilon \in (0, 1]$.

▶ **Theorem 35.** *Let $\mathcal{S}^*_{\mathbb{F},\epsilon}$ denote an SMT of $X'(\mathbb{F}, \epsilon)$ and $|\mathcal{S}^*_{\mathbb{F},\epsilon}|$ denote its length. If $\mathbb{F}$ has an exact cover, then $|\mathcal{S}^*_{\mathbb{F},\epsilon}| \leq f(n, t, \hat{C}) + |\mathcal{T}_1|$, otherwise $|\mathcal{S}^*_{\mathbb{F},\epsilon}| \geq f(n, t, \hat{C}) + \frac{1}{200nt} + |\mathcal{T}_1|$, where $\hat{C}$ is the number of crossovers, i.e. hexagonal gadgets, and f is a positive real-valued function of $n, t, \hat{C}$ as stated in Proposition 26.*

Since it is not known if the ESMT problem is in NP, Garey et al. [6] show the NP-completeness of a related problem called the Discrete Euclidean Steiner Minimal Tree (DESMT) problem, which is in NP. We define the DESMT problem as given in [6]. The DESMT problem takes as input a set $\mathcal{X}$ of integer-coordinate points in the plane and a positive integer L, and asks if there exists a set $\mathcal{Y} \supseteq \mathcal{X}$ of integer-coordinate points such that some spanning tree $\mathcal{T}$ for $\mathcal{Y}$ satisfies $|\mathcal{T}|_d \leq L$, where $|\mathcal{T}|_d = \Sigma_{e \in E(\mathcal{T})} \lceil \overline{e} \rceil$, i.e. we round up the length of each edge to the least integer not less than it.

In order to show that DESMT is NP-hard, the same reduction as that of the ESMT problem can be used, followed by scaling and rounding the coordinates of the points. Theorem 4 of [6] proves that the DESMT problem is NP-Complete. Moreover, since it is Strongly NP-Complete, the DESMT problem does not admit any FPTAS. Finally in Theorem 5 of [6], Garey et al. show that as a consequence, the ESMT problem does not have any FPTAS as well.

Now we show that the DESMT problem is NP-hard even on $f(n)$-Almost Convex Sets of n terminals, when $f(n) = \Omega(n^\epsilon)$ and where $\epsilon \in (0, 1]$.

In Section 7 of [6], the reduced instance $X(\mathbb{F})$ of ESMT is converted into an instance $X_d(\mathbb{F})$ of DESMT. The conversion is as follows:
$X_d(\mathbb{F}) = \{(\lceil 12M \cdot 200nt \cdot x_1 \rceil, \lceil 12M \cdot 200nt \cdot x_2 \rceil) : x = (x_1, x_2) \in X(\mathbb{F})\}$, where $M = |X(\mathbb{F})|$.

We apply a similar conversion to the reduced ESMT instance $X'(\mathbb{F}, \epsilon)$, to convert it into a DESMT instance of an $\Omega(n^\epsilon)$-Almost Convex Set. The conversion goes as follows:
$X'_d(\mathbb{F}, \epsilon) = \{(\lceil 12N \cdot 200nt \cdot x_1 \rceil, \lceil 12N \cdot 200nt \cdot x_2 \rceil) : x = (x_1, x_2) \in X'(\mathbb{F}, \epsilon)\}$, where $N = |X'(\mathbb{F}, \epsilon)|$.

The next two lemmas establish the validity of $X'_d(\mathbb{F}, \epsilon)$ as an instance of DESMT and the upper bounds on the size of the constructed instance. Note that the reduction from X3C followed by the conversion can be done in polynomial time.

▶ **Lemma 36.** *The instance $X'_d(\mathbb{F}, \epsilon)$ constructed above is a valid DESMT instance.*

▶ **Lemma 37.** *The reduced DESMT instance $X'_d(\mathbb{F}, \epsilon)$ has N distinct points, where $N = |X'(\mathbb{F}, \epsilon)|$.*

Now we present the following lemma for the constructed DESMT instance $X'_d(\mathbb{F}, \epsilon)$ analogous to Lemma 29 for the ESMT instance $X'(\mathbb{F}, \epsilon)$.

▶ **Lemma 38.** *The reduced DESMT instance $X'_d(\mathbb{F}, \epsilon)$ constructed is an $\Omega(N^\epsilon)$-Almost Convex Set, where $N = |X'_d(\mathbb{F}, \epsilon)|$.*

We get the following theorem from Lemmas 36–38.

▶ **Theorem 39.** *The instance $X'_d(\mathbb{F}, \epsilon)$ constructed is a valid DESMT instance on an $\Omega(N^\epsilon)$-Almost Convex Set, where $|X'_d(\mathbb{F}, \epsilon)| = |X'(\mathbb{F}, \epsilon)| = N$.*

Following Theorems 3 and 4 in [6], we get that the DESMT problem is NP-Complete for $\Omega(N^\epsilon)$-Almost Convex Sets, where N is the total number of terminals. Since we get the reduced instance $X'_d(\mathbb{F}, \epsilon)$ from the X3C instance $(n, \mathbb{F})$, the DESMT problem is strongly NP-complete for $\Omega(N^\epsilon)$-Almost Convex Sets, and does not admit any FPTAS.

Using Theorem 5 of [6], we get that if the ESMT problem has an FPTAS, then the X3C problem can be solved in polynomial time. The Theorem also applies for our case of $\Omega(N^\epsilon)$-Almost Convex Sets. Therefore, we get the following theorem,

▶ **Theorem 40.** *There does not exist any FPTAS for the ESMT problem on $f(n)$-Almost Convex Sets of n terminals, where $f(n) = \Omega(n^\epsilon)$ and $\epsilon \in (0, 1]$, unless $P = NP$.*

6 Conclusion

In this paper, we first study ESMT on vertices of 2-CPR n-gons and design a polynomial time algorithm. It remains open to design a polynomial time algorithm for ESMT on k-CPR n-gons, or show NP-hardness for the problem. Next, we study the problem on $f(n)$-Almost Convex Sets of n terminals. For this NP-hard problem, we obtain an algorithm that runs in $2^{\mathcal{O}(f(n)\log n)}$ time. We also design an FPTAS when $f(n) = \mathcal{O}(\log n)$. On the other hand, we show that there cannot be an FPTAS if $f(n) = \Omega(n^\epsilon)$ for any $\epsilon \in (0, 1]$, unless P = NP. The question of existence of FPTASes when $f(n)$ is a polylogarithmic function remains open.

References

1 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.

2 Marcus Brazil, Ronald L Graham, Doreen A Thomas, and Martin Zachariasen. On the history of the Euclidean Steiner tree problem. *Archive for history of exact sciences*, 68(3):327–354, 2014.

3 EJ Cockayne. On the Steiner problem. *Canadian Mathematical Bulletin*, 10(3):431–450, 1967.

4 Stuart E Dreyfus and Robert A Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

5 Ding-Zhu Du, Frank K. Hwang, and JF Weng. Steiner minimal trees for regular polygons. *Discrete & Computational Geometry*, 2(1):65–84, 1987.

6 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.

7 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

8 FK Hwang. A linear time algorithm for full Steiner trees. *Operations Research Letters*, 4(5):235–237, 1986.

9 FK Hwang, DS Richards, and P Winter. The Steiner tree problem. Annals of Discrete Mathematics series, vol. 53, 1992.

10 Sándor Kisfaludi-Bak, Jesper Nederlof, and Karol Węgrzycki. A gap-ETH-tight approximation scheme for Euclidean TSP. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 351–362. IEEE, 2022.

11 Zdzislaw Alexander Melzak. On the problem of Steiner. *Canadian Mathematical Bulletin*, 4(2):143–148, 1961.

12 Satish B Rao and Warren D Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 540–550, 1998.

13 J Hyam Rubinstein, Doreen A Thomas, and Nicholas C Wormald. Steiner trees for terminals constrained to curves. *SIAM Journal on Discrete Mathematics*, 10(1):1–17, 1997.

14 J Scott Provan. Convexity and the Steiner tree problem. *Networks*, 18(1):55–72, 1988.

15 Jia Feng Weng and Raymond Sydney Booth. Steiner minimal trees on regular polygons with centre. *Discrete Mathematics*, 141(1-3):259–274, 1995.

Rectilinear-Upward Planarity Testing of Digraphs

Walter Didimo
Department of Engineering, University of Perugia, Italy

Michael Kaufmann
Department of Computer Science, University of Tübingen, Germany

Giuseppe Liotta
Department of Engineering, University of Perugia, Italy

Giacomo Ortali
Department of Engineering, University of Perugia, Italy

Maurizio Patrignani
Department of Civil, Computer and Aeronautical Engineering, Roma Tre University, Italy

Abstract

A *rectilinear-upward planar drawing* of a digraph G is a crossing-free drawing of G where each edge is either a horizontal or a vertical segment, and such that no directed edge points downward. Rectilinear-Upward Planarity Testing is the problem of deciding whether a digraph G admits a rectilinear-upward planar drawing. We show that: (i) Rectilinear-Upward Planarity Testing is NP-complete, even if G is biconnected; (ii) it can be solved in linear time when an upward planar embedding of G is fixed; (iii) the problem is polynomial-time solvable for biconnected digraphs of treewidth at most two, i.e., for digraphs whose underlying undirected graph is a series-parallel graph; (iv) for any biconnected digraph the problem is fixed-parameter tractable when parameterized by the number of sources and sinks in the digraph.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Dynamic programming; Theory of computation → Graph algorithms analysis

Keywords and phrases Graph drawing, orthogonal drawings, upward drawings, rectilinear planarity, upward planarity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.26

Funding *Walter Didimo*: Project AIDMIX – Artificial Intelligence for Decision Making: Methods for Interpretability and eXplainability – Ricerca di Base 2021.
Giuseppe Liotta: MUR of Italy, PRIN Project n. 2022TS4Y3N – EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data.

Acknowledgements We thank Ignaz Rutter for conversations about the problem 1-2-Switch-Flow.

1 Introduction

A *rectilinear planar drawing* of a graph G is a crossing-free drawing of G where vertices are placed at distinct points in the plane (possibly at grid points) and edges are drawn as either horizontal segments or vertical segments. Rectilinear Planarity Testing is the problem of deciding whether a planar graph admits a rectilinear planar drawing. Besides the theoretical beauty of the problem, which belongs to the vast literature about graph planarity testing (see, e.g. [10, 38, 39] for books and surveys), the question is at the heart of those technologies that display networked data by means of orthogonal layouts, which find applications in a variety of fields, from software engineering to bioinformatics, from data bases to computer networks (see, e.g., [21, 36]).

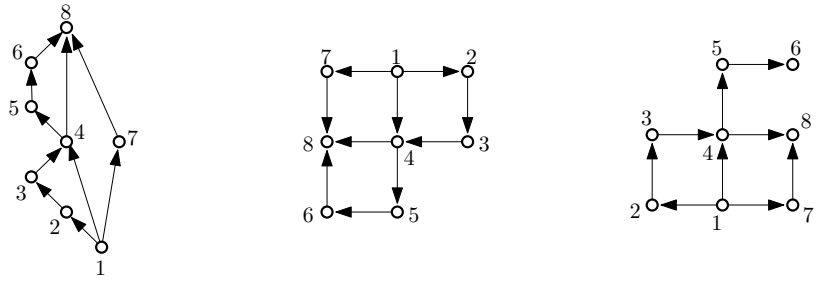
Rectilinear Planarity Testing has been proved to be NP-complete [29]. However, polynomial-time solutions are known both for constrained versions of the problem and for restricted families of graphs. Namely, Rectilinear Planarity Testing can be solved in polynomial time in the so-called *fixed-embedding setting*, that is when the input is given with a planar embedding and the testing algorithm is not allowed to change the embedding (see, e.g. [42]). Also, polynomial-time solutions are known for graphs of bounded treewidth and for sub-cubic graphs (see, e.g., [11, 12, 13, 22, 27, 31, 40, 41]).

In this paper we study rectilinear planar drawings of directed graphs (digraphs). We want to test whether a digraph G admits a rectilinear planar drawing with the additional constraint that no directed edge points downward. We call such a drawing a *rectilinear-upward planar drawing* and the testing problem Rectilinear-Upward Planarity Testing. It may be worth recalling that the problem of testing whether a digraph admits an *upward planar drawing*, i.e., a planar drawing where each edge is monotonically increasing in the upward direction according to its orientation, is NP-complete [29]. See also [1, 3, 4, 5, 7, 15, 32, 34] for polynomial-time solutions and parameterized approaches on restricted graph families or scenarios. Figure 1 shows an example of a digraph that admits both an upward planar drawing and a rectilinear planar drawing, but that does not admit a rectilinear-upward planar drawing. Our contributions can be summarized as follows.

- We prove that Rectilinear-Upward Planarity Testing is NP-complete, even if the input digraph is biconnected (Section 3).
- We show that Rectilinear-Upward Planarity Testing can be solved in linear-time when an upward planar embedding of G is fixed as part of the input (Section 4). We remark that both the problem of testing rectilinear planarity and of testing upward planarity in linear time in the fixed-embedding setting are among of the most famous and long-standing open problems in graph drawing (see, e.g., [6, 43]).
- We consider the variable-embedding setting, where the algorithm is free to chose the planar embedding of the input graph, and we focus on families of biconnected digraphs (Section 5). We show that Rectilinear-Upward Planarity Testing can be solved in polynomial time for biconnected digraphs with treewidth at most two, i.e., when the underlying undirected graph is series-parallel. We recall that polynomial-time testing algorithms for series-parallel graphs are known in the literature both in the context of upward planarity testing only and in the context of rectilinear planarity testing only (see, e.g., [8, 19, 16]). We also show that Rectilinear-Upward Planarity Testing is FPT when parameterized by the number k of sources and sinks of the digraph. Namely, for any n-vertex digraph our FPT algorithm is single-exponential in k and has a quadratic factor in n. We remark that parameterized complexities of upward and rectilinear planarity testing are topics that have been receiving increasing attention (see, e.g., [7, 14, 35]).

From a technical point of view, our linear-time algorithm in the fixed-embedding setting exploits a 2-SAT formulation, instead of using network-flow models as done in the standard approaches for testing both rectilinear and upward planarity (see, e.g., [1, 3, 9, 28, 42]). In the variable-embedding setting, we rely on the concept of *rectilinear-upward spirality*. It combines the notion of spirality introduced in [11] to measure how much a triconnected component of a rectilinear drawing can be "rolled up", with additional information about the orientation of the edges incident to the poles of the triconnected components.

For space restrictions some proofs are sketched or omitted. Full proofs will appear in an extended journal version of the paper.



(a) (b) (c)

Figure 1 A graph G that is upward planar, rectilinear planar, but not rectilinear-upward planar. (a) An upward planar drawing of G. (b) A rectilinear planar drawing of G. (c) A rectilinear-upward planar drawing of G without edge $(6, 8)$.

2 Basic Definitions and Properties

For basic definitions on graph drawing and planarity refer to [10]. We assume to work with connected graphs, as otherwise we can treat each connected component of the graph independently. A 4-*graph* is a graph with vertex-degree at most four.

Upward planar drawings. In an *upward drawing* of a digraph G each edge is represented as a Jordan arc monotonically increasing in the upward direction, according to its orientation; see Figure 1a. A digraph G is *upward planar* if it admits an upward planar drawing. Clearly, a necessary (but not sufficient) condition for G to be upward planar is that G is acyclic.

Orthogonal drawings and representations. Let G be a planar (undirected or directed) 4-graph, and let Γ be a planar drawing of G. We say that Γ is an *orthogonal planar drawing* of G if each edge is drawn as a sequence of horizontal and vertical segments. A *bend* on an edge is the contact point between a horizontal and a vertical segment of the edge. An *orthogonal planar representation* H of a planar graph G is a class of shape equivalent orthogonal planar drawings; namely, H describes the planar embedding of G, the sequence of left/right bends along the edges, and the angles at every vertex of G, each angle formed by two (possibly coincident) consecutive edges around the vertex and expressed as a value in the set $\{90°, 180°, 270°, 360°\}$. If H is the orthogonal representation of an orthogonal planar drawing Γ, we also say that Γ *preserves* H and that Γ is a *drawing of* H. A drawing of H can be computed in linear time [42], thus we can concentrate on computing orthogonal representations rather than drawings.

Orthogonal planar drawings (resp. representations) without bends are called *rectilinear planar drawings* (resp. *rectilinear planar representations*); see, e.g., Figure 1b. A graph G is *rectilinear planar* if it admits a rectilinear planar drawing (or representation). We say that a rectilinear planar representation H is *oriented* if it also specifies for each edge (u, v) of G the relative position of u with respect to v, i.e., whether u must be to the left, to the right, above, or below v in every rectilinear drawing of H; in particular, this information establishes for each edge e of G if e is horizontal or vertical in H (it is actually enough to specify the relative position of the end-vertices of one edge of H to establish the relative position of the end-vertices for every other edge). Note that the definition of oriented rectilinear representation H of G has nothing to do with the orientation of the edges when G is a digraph. For a given rectilinear representation of G there are always four different oriented versions of it, obtained by rotating one of them by an angle of $k \cdot 90°$, for $k = 0, 1, 2, 3$.

Rectilinear-upward planar representations. In this paper we deal with drawings of digraphs that are at the same time rectilinear and upward. More precisely, we do not require that an edge is strictly upward (which would prevent us from drawing it as a horizontal segment), but rather we exclude that it is drawn downward. Formally, let G be an acyclic planar 4-digraph and let Γ be a planar drawing of G. We say that Γ is a *rectilinear-upward planar drawing* of G if Γ is a rectilinear planar drawing of G with no directed edge that points downward; see, e.g., Figure 1c. This corresponds to saying that a rectilinear upward planar drawing Γ induces an oriented rectilinear planar representation H of G with the property that for each directed edge (u, v) of G, vertex u is never above vertex v. We say that H is a *rectilinear-upward planar representation* of G. As for rectilinear representations, H describes a class of shape equivalent rectilinear-upward planar drawings. A digraph G is *rectilinear-upward planar* if it admits a rectilinear-upward planar drawing, or equivalently, a rectilinear-upward planar representation. Clearly, rectilinear planarity is necessary for rectilinear-upward planarity. The next property implies that also upward planarity is necessary for rectilinear-upward planarity. As already observed, both rectilinear planarity and upward planarity are not sufficient conditions when considered independently (see Figure 1).

▶ **Property 1.** *If Γ is a rectilinear-upward planar drawing of a digraph G, then Γ can be transformed into an upward planar drawing of G with the same planar embedding as Γ.*

In the remainder we only consider planar 4-digraphs, thus we often omit the term "planar" and we avoid to specify that the vertex-degree is at most four. Also, we often use the abbreviation "RU" in place of "rectilinear-upward". Finally, we implicitly assume that the input digraphs are acyclic, as otherwise an upward drawing, and hence an RU drawing, cannot exist. Acyclicity can be tested in linear time, by a classical depth first search.

3 NP-Completeness of Rectilinear-Upward Planarity Testing

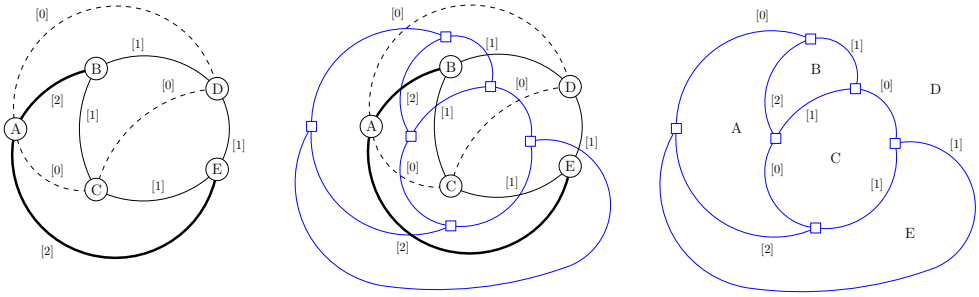
To prove the hardness of Rectilinear-Upward Planarity Testing, we use a reduction from the following 1-2-Switch-Flow problem, introduced in this paper and that may be considered of independent interest. The hardness of 1-2-Switch-Flow can be proved with a reduction from the problem Flow Orientation, which is shown to be NP-complete even if edge capacities are $O(\sqrt{n})$, where n is the number of vertices of the graph [23].

Problem: 1-2-Switch-Flow (12SW)

Instance: A planar undirected graph $G = (V, E)$ where each edge $e \in E$ is labeled with a value $f_e \in \{1, 2\}$.

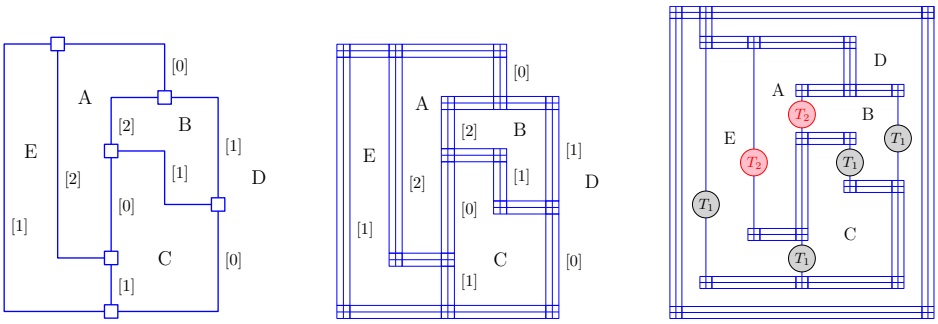
Question: Does there exist an orientation for the edges in E such that for each vertex the sum of the values of the incoming edges equals the sum of values of the outgoing edges?

We now sketch the reduction from 1-2-Switch-Flow (12SW) to Rectilinear-Upward Planarity Testing. Let $G = (V, E)$ be an instance of 12SW. We first compute a planar embedding of G and planarly add extra edges with label [0] (called [0]-*edges*) in such a way to obtain a maximal plane graph G^+ (see Figure 2a). Second, we compute the dual plane graph G^* of G^+ and label the edges of G^* with the values of the corresponding edges of G^+ (see Figures 2b and 2c). Third, we compute an orthogonal drawing Γ_{G^*} of G^* such that each edge has at least one vertical segment (see Figure 3a). Fourth, we transform Γ_{G^*} into an auxiliary positive instance F of Rectilinear-Upward Planarity Testing by replacing orthogonal and vertical segments with rectangular boxes. In particular, each edge



(a) Graph G^+. **(b)** Graph G^+ and its dual G^*. **(c)** Graph G^*.

Figure 2 The first two steps of the reduction from 12SW to Rectilinear-Upward Planarity Testing.



(a) Drawing Γ_{G^*} of G^*. **(b)** Instance F. **(c)** Instance I_{RUPT}.

Figure 3 The last steps of the reduction from 12SW to Rectilinear-Upward Planarity Testing.

segment of Γ_{G^*} is replaced with three parallel edges and each vertex or bend is replaced with a 3×3 grid (see Figure 3b). Edges of the instance F are oriented so that F admits a unique rectilinear-upward planar representation $\mathcal{H}_F$ up to a horizontal flip. Fifth, for each edge e of G^* labeled [1] (labeled [2], respectively), we identify the three parallel edges of F corresponding to a vertical segment of Γ_{G^*} belonging to e and replace them with the subgraph T_1 (the subgraph T_2, respectively). Subgraphs T_1 and T_2, called *tendrils*, are depicted in Figure 7a and Figure 7b, respectively. Finally, we add a framework all around the graph and attach it to a vertex or bend in the upper part of Γ_{G^*} (as in Figure 3c) to obtain the desired instance I_{RUPT} of Rectilinear-Upward Planarity Testing.

▶ **Theorem 1.** *Rectilinear-Upward Planarity Testing is NP-complete.*

Sketch of Proof. The problem is in NP since the recognition of an RU representation is polynomial-time solvable. As for the hardness, we show that the above described reduction from 1-2-Switch-Flow constructs an instance I_{RUPT} of Rectilinear-Upward Planarity Testing that admits an RU representation if and only if its tendrils are embedded in such a way that, for each face of Γ_{G^*}, the number of extra 270° angles provided by some tendrils equals the number of extra 90° angles provided by the remaining tendrils of the same face and this is equivalent to finding a feasible flow for the original 1-2-Switch-Flow instance. ◀

4 Testing Upward Plane Digraphs in Linear Time

If we have in input a rectilinear planar representation H of a digraph G, testing whether H is also an RU representation for one of the four possible orientations of H is a trivial problem. On the contrary, given a plane graph G with a prescribed "upward planar embedding", testing whether it admits an RU representation is a relevant problem. In this section we address this problem and present a linear-time algorithm to test whether an *upward plane* digraph G admits an RU representation. We recall that an early paper in the graph drawing literature [26] claims the result of this section. Unfortunately, that paper only gives a sketch about the algorithm to test RU planarity without giving sufficient details and simultaneously referring to a much more restrictive model [25].

Before describing our algorithm, we formalize the concept of upward plane digraph. In any RU representation H of a digraph G each vertex v is *bimodal*, i.e., all the incoming edges of v (as well as all the outgoing edges of v) are consecutive around v. More specifically, H induces: (a) a planar embedding of G and, (b) for each vertex v of G, a linear left-to-right (possibly empty) list of the incoming edges of v and a linear left-to-right (possibly empty) list of the outgoing edges of v. The information (a) and (b) together are called an *upward planar embedding* of G. A digraph G is upward plane if it comes with a given upward planar embedding. An RU representation of an upward plane digraph G (if any) is an RU representation of G that preserves its upward planar embedding. Note that, given information (a) and (b) for a planar digraph G, it can be easily checked in linear time whether this pair correctly defines an upward plane embedding, i.e., if there exists an upward planar drawing of G whose upward plane embedding coincides with (a) and (b) (see e.g. [3]).

The main ingredient of our approach is a 2-SAT formulation of the testing problem. It consists of three phases, summarized hereunder and then described in more detail.

- Phase 1: For each vertex w and for each edge e outgoing w, we assign a set $\lambda_{\text{out}}(e)$ of labels to e, encoding the sides by which e can leave w. Each of these labels is chosen in the set $\{E, W, N\}$ (East, West, or North). Similarly, for each edge e incoming w, we assign to e a set $\lambda_{\text{in}}(e)$ of $\{E, W, S\}$ (East, West, or South), encoding the sides by which e can enter w.
- Phase 2: Based on $\lambda_{\text{out}}(e)$ and $\lambda_{\text{in}}(e)$ for each directed edge $e = (u, v)$, we compute a set $\lambda(e)$ of labels, each label taken in the set $\{L, U, R\}$, such that $|\lambda(e)| \leq 2$. The set $\lambda(e)$ encodes the possible directions (L=leftward, U=upward, R=rightward, respectively) that an edge can have in an RU representation. If $\lambda(e)$ is an empty set then the input graph does not have an RU representation. The function λ is a *candidate set of labels* for the edges of G.
- Phase 3: By exploiting the labels associated with the edges, RU planarity is modeled as a 2-SAT formula ϕ, which is then solved in linear time [37].

Details for Phase 1. We describe how to define the sets $\lambda_{\text{out}}(\cdot)$ and $\lambda_{\text{in}}(\cdot)$ for every edge outgoing or incoming a vertex w of G. (i) If w has three outgoing (resp. incoming) edges e_1, e_2, e_3, in this left-to-right order in the upward planar embedding of G, then the sides from which these edges are incident to w can be uniquely fixed. Namely, $\lambda_{\text{out}}(e_1) = \{W\}$, $\lambda_{\text{out}}(e_2) = \{N\}$, $\lambda_{\text{out}}(e_3) = \{E\}$ (resp. $\lambda_{\text{in}}(e_1) = \{W\}$, $\lambda_{\text{in}}(e_2) = \{S\}$, $\lambda_{\text{in}}(e_3) = \{E\}$). (ii) If w has two outgoing (resp. incoming) edges e_1 and e_2, in this left-to-right order, we set $\lambda_{\text{out}}(e_1) = \{W, N\}$, $\lambda_{\text{out}}(e_2) = \{N, E\}$ (resp. $\lambda_{\text{in}}(e_1) = \{W, S\}$, $\lambda_{\text{in}}(e_2) = \{S, E\}$). (iii) If

w has one outgoing edge e_1 we set $\lambda_{\text{out}}(e_1) = \{W, N, E\}$ in all cases except when w has three incoming edges, in which case $\lambda_{\text{out}}(e_1) = \{N\}$. (*iv*) If w has one incoming edge e_1 we set $\lambda_{\text{in}}(e_1) = \{W, S, E\}$ in all cases except when w has three outgoing edges, in which case $\lambda_{\text{in}}(e_1) = \{S\}$.

Details for Phase 2. For each edge e, given the label sets $\lambda_{\text{out}}(e)$ and $\lambda_{\text{in}}(e)$ for e, we first initialize $\lambda(e)$ as the empty set. If $S \in \lambda_{\text{in}}(e)$ and $N \in \lambda_{\text{out}}(e)$, we add label U to $\lambda(e)$. If $E \in \lambda_{\text{out}}(e)$ and $W \in \lambda_{\text{in}}(e)$, we add label R to $\lambda(e)$. If $W \in \lambda_{\text{out}}(e)$ and $E \in \lambda_{\text{in}}(e)$, we add label L to $\lambda(e)$. We say that λ is a *good labeling* if it exists an RU representation H of G such that each edge $e \in H$ has a direction that corresponds to one of the labels of $\lambda(e)$; if so, H is said to be *compatible* with λ. Note that the labeling λ constructed as described above is such that, for each edge $e = (u, v)$, $|\lambda(e)| \leq 3$. Also, if $|\lambda(e)| = 3$ then $\lambda(e) = \{L, U, R\}$, and e is the only outgoing edge of u and the only incoming edge of v. Consider another labeling λ', derived from λ as follows: If $|\lambda(e)| \leq 2$, let $\lambda'(e) = \lambda(e)$; if $|\lambda(e)| = 3$, let $\lambda'(e) = \{U\}$. Clearly, λ' is constructed in linear time from λ and $|\lambda'(e)| \leq 2$, for every edge e of the graph. We call λ' the *reduction of* λ. The following lemma is crucial for our 2-SAT model.

▶ **Lemma 2.** *λ is a good labeling if and only if its reduction λ' is a good labeling.*

Proof. Clearly, if λ' is a good labeling then λ is, because $\lambda(e)$ is a superset of $\lambda'(e)$. Suppose, vice versa, that λ is a good labeling. We prove that λ' is a good labeling by induction on the number k of edges e for which $|\lambda(e)| = 3$. If $k = 0$, λ and λ' coincides, and the statement is obvious. Suppose that the statement is true for any $k \geq 1$, and let e be any edge for which $|\lambda(e)| = 3$. Let λ'' be the labeling obtained from λ' by setting $\lambda''(e) = \lambda(e) = \{L, U, R\}$. By the inductive hypothesis λ'' is a good labeling. Consider an RU representation H of G compatible with λ and let d be the direction of e in H. If d is the upward direction, then H is also compatible with λ', because $\lambda'(e) = \{U\}$. Otherwise (i.e., d is either the rightward or the leftward direction) there is neither an edge of H that leaves u from North nor an edge of H that enters v from South (because e is the only outgoing edge of u and the only incoming edge of v). Hence, we can derive from H another RU representation H' such that e points upward while all other edges of H' have the same direction as in H. The representation H' is now compatible with λ', which implies that λ' is a good labeling. ◀

Details for Phase 3. By Lemma 2, we can always assume that the labeling λ determined in the previous phase is such that $\lambda(e)$ contains either one or two labels, for each edge e of G. Indeed, if this is not the case, we can restrict to consider its reduction λ', obtained from λ in linear time. Let w be a vertex of G and let e_1 and e_2 be two edges of G that are either both outgoing w or both incoming w. Two labels $X \in \lambda(e_1)$ and $Y \in \lambda(e_2)$ are *conflicting* if $X = Y$. This is true, because there cannot exist an RU representation of G such that the directions of e_1 and e_2 coincide. Let e_1 be an edge outgoing w and let e_2 be an edge incoming w. Two labels $X \in \lambda(e_1)$ and $Y \in \lambda(e_2)$ are *conflicting* if X and Y represent *opposite directions* (i.e., $X = L$ and $Y = R$ or $X = R$ and $Y = L$). This phase aims to assign a single label to each edge, in such a way that there is no conflicting labels. Such an assignment (if any) is a *non-conflicting label assignment within* λ. We use the notation $\mathcal{L}(\lambda)$ to denote any non-conflicting assignment within λ. The next lemma establishes an equivalence between non-conflicting label assignments and RU representations of G compatible with λ.

▶ **Lemma 3.** *Let λ be a candidate set of labels for the edges of G. There exists an RU representation H that is compatible with λ if and only if there exists a non-conflicting label assignment within λ. The edge directions defined by H correspond to those defined by the label assignment, and H preserves the planar embedding of G.*

Proof. If there exists an RU representation H that is compatible with λ, then choosing for each edge e the label of $\lambda(e)$ that corresponds to the direction of e in H immediately yields a non-conflicting label assignment within λ.

Suppose vice versa that there exists a non-conflicting label assignment $\mathcal{L}(\lambda)$. We show that from $\mathcal{L}(\lambda)$ we can derive an RU representation H of G that is compatible with λ. Since by hypothesis G is upward planar and comes with an upward-planar embedding, there exists a straight-line upward planar drawing Γ' of G that preserves its upward planar embedding [2]. We can construct from Γ' an orthogonal-upward drawing Γ'' of G such that for each edge $e = (u, v)$: (i) the directions of the segments of e that are incident to u and v are coherent with the label of e in $\mathcal{L}(\lambda)$; and (ii) moving from u to v along e, the number of right bends equals the number of left bends.

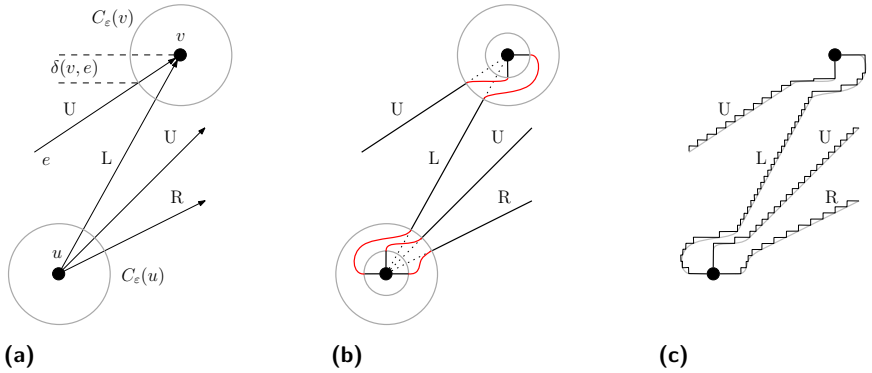


Figure 4 An illustration for the proof of Lemma 3.

To construct Γ'', proceed as follows. Let $\varepsilon > 0$ be a length such that the circular area of radius ε around each vertex v does not intersect any other vertex and any other edge that is not incident to v in Γ'. For each vertex v of Γ', draw a circle $C_\varepsilon(v)$, centered at v, of radius ε (see Figure 4a). Let e be an edge incident to v. Denote by $\delta(v, e)$ the smallest vertical distance between v and the intersection of e with $C_\epsilon(v)$ (see Figure 4a). Let δ be the minimum of all $\delta(v, e)$. Draw a circle $C_\delta(v)$ of radius δ around each vertex v. Now construct a drawing of G such that each edge $e = (u, v)$ is non-decreasing with respect to the y-coordinate, leaves the u vertex and enters vertex v with a straight segment of length δ directed as prescribed by λ. This is possible because $\delta = \min_{v,e}\{\delta(v, e)\}$ (see Figure 4b). To finally obtain Γ'', we replace each edge e by a sequence of horizontal and vertical segments that follows the drawing of e at a distance that is small enough to guarantee that it does not intersect any other edge or vertex of the drawing (see Figure 4a). Since the sequence of horizontal and vertical segments of each edge e starts and ends with a segment that goes in the same direction and is non-decreasing, the numbers of right and left turns are the same.

To construct the final RU representation H, consider the orthogonal representation H'' of Γ''. Since each edge of H'' has the same number of left and right turns, by [42] there exists an orthogonal representation H of G without bends (i.e., a rectilinear representation of G) such that H has the same embedding as H'' and such that each edge in H is incident to its end-vertices from the same side as in H''. ◀

Given a non-conflicting label assignment $\mathcal{L}(\lambda)$ of G, an RU representation H of G compatible with λ and whose edge directions correspond to the edge labels of $\mathcal{L}(\lambda)$, can be easily constructed in linear time. Namely, since H preserves the planar embedding of G,

for each vertex w of H the angles at w can be easily determined by the label assignment $\mathcal{L}(\lambda)$ for the edges incident to w. Also, H is oriented in such a way that the directions of the edges are coherent with λ. We now give the main result of this section.

▶ **Theorem 4.** *Let G be an n-vertex upward plane digraph. There exists an $O(n)$-time algorithm that tests whether G admits an RU representation, and that computes one in the positive case.*

Proof. Let λ be a candidate set of labels for the edges of G, computed as described in Phase 1 and Phase 2. By Lemma 2, we also assume that, for each edge e of G, $\lambda(e)$ contains at most two labels. Based on Lemma 3, deciding whether G admits an embedding-preserving RU representation is equivalent to deciding whether G admits a non-conflicting labeling $\mathcal{L}(\lambda)$. We model this problem as a 2-SAT problem, which is defined as follows.

For each edge e and for each label $X \in \lambda(e)$, define a Boolean variable b_e^X; this variable will be set to `True` if we select label X for edge e, and it will be set to `False` otherwise. We define a formula $cl(e)$ for every edge e and a formula $cl(v)$ for every vertex v that has at least one incident edge e with $|\lambda(e)| = 2$. Our 2-SAT formula Φ is the conjunction of all the formulas defined for the edges and for the vertices of G.

For each edge e of G we define $cl(e)$ as either the conjunction of two clauses or as a single clause in Φ, depending on whether $|\lambda(e)| = 2$ or $|\lambda(e)| = 1$. More precisely, if $\lambda(e) = \{X, Y\}$ we have $cl(e) = (b_e^X \vee b_e^Y) \wedge (\neg b_e^X \vee \neg b_e^Y)$. This ensures that in order to satisfy Φ we have to select exactly one of the two labels X and Y. If $\lambda(e) = \{X\}$, we have $cl(e) = (b_e^X \vee b_e^X)$. For $cl(v)$ we have two cases: (i) v is a source or a sink; (ii) v is neither a source nor a sink.

Case (i). If v is a source (resp. a sink), let e_1 and e_2 be the two outgoing (resp. incoming) edges from v, respectively. We set:

$$cl(v) = \neg b_{e_1}^U \vee \neg b_{e_2}^U$$

Case (ii). We have four subcases: (a) $\deg(v) = 2$; (b) $\deg(v) = 3$ and v has two incoming edges; (c) $\deg(v) = 3$ and v has two outgoing edges; (d) $\deg(v) = 4$ and v has two incoming and two outgoing edges.

(a) Let e_1 and e_2 be the incoming edge and the outgoing edge of v, respectively. Suppose $\lambda(e_1) = \{U, X\}$ and $\lambda(e_2) = \{U, Y\}$, where $X, Y \in \{R, L\}$. If $X = Y$, we do not add any clause associated with v, because any two labels for e_1 and e_2 in $\lambda(e_1)$ and in $\lambda(e_2)$ are non-conflicting. If $X \neq Y$, we set:

$$cl(v) = \neg b_{e_1}^X \vee \neg b_{e_2}^Y.$$

(b) Let e_1 and e_2 be the two incoming edges of v, in this left-to-right order, and let e_3 be the outgoing edge of v. We have $\lambda(e_1) = \{U, R\}$, $\lambda(e_2) = \{L, U\}$, and either (1) $\lambda(e_3) = \{L, U\}$ or (2) $\lambda(e_3) = \{U, R\}$. We define $cl(v)$ as follows, depending on the two sub-cases:

(1) $cl(v) = (\neg b_{e_1}^U \vee \neg b_{e_2}^U) \wedge (\neg b_{e_1}^R \vee \neg b_{e_3}^L)$

(2) $cl(v) = (\neg b_{e_1}^U \vee \neg b_{e_2}^U) \wedge (\neg b_{e_2}^L \vee \neg b_{e_3}^R)$

(c) Symmetric to (b).

(d) Let e_1 and e_2 be the two outgoing edges of v, and let e_3 and e_4 be the two incoming edges of v, in this left-to-right order. We have: $\lambda(e_1) = \{L, U\}$, $\lambda(e_2) = \{U, R\}$; $\lambda(e_3) = \{L, U\}$, $\lambda(e_4) = \{U, R\}$. We define $cl(v)$ as follows:

$$cl(v) = (\neg b_{e_1}^U \vee \neg b_{e_2}^U) \wedge (\neg b_{e_2}^R \vee \neg b_{e_3}^L) \wedge (\neg b_{e_3}^U \vee \neg b_{e_4}^U) \wedge (\neg b_{e_1}^L \vee \neg b_{e_4}^R)$$

Observe that, if a vertex is incident to an edge e with $|\lambda(e)| = 1$, we use the clauses defined above, but in these cases they are simplified since the values b_e^X are fixed (either to True or to False) for any possible value of X. ◀

5 Testing in the Variable Embedding Setting

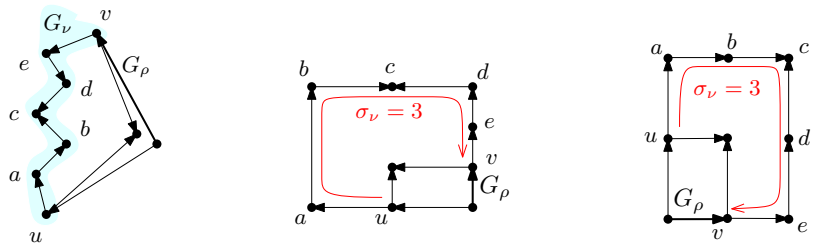
In this section we deal with biconnected planar digraphs whose embedding is not fixed. In Section 5.1 we define the notion of rectilinear-upward spirality. In Section 5.2 we describe a polynomial-time testing algorithm for digraphs whose underlying undiracted graph is series-parallel. In Section 5.3 we consider the general case, and give a FPT algorithm parameterized by the the number of sources and sinks in the digraph.

5.1 Rectilinear-Upward Spirality

We introduce the new concept of rectilinear-upward spirality, which specializes the notion of orthogonal spirality defined in [11]. While the orthogonal spirality is a measure of how much a given subgraph of an undirected graph G is rolled-up in an orthogonal representation of G, our notion of spirality is for directed graphs and incorporates additional information about the sides to which edges are incident to the poles of the triconnected components.

SPQR-trees. As in [11], our definition of spirality exploits the popular SPQR-tree data structure introduced by Di Battista and Tamassia [10]. The *SPQR-tree* T of a biconnected (di)graph G represents the decomposition of G into its triconnected components [33], and it can be computed in linear time [10, 30]. Refer to Figure 8. Each triconnected component corresponds to a non-leaf node ν of T; the triconnected component itself is the *skeleton* of ν and is denoted as skel(ν). Node ν can be: (*i*) an *S-node* (series composition), if skel(ν) is a simple cycle of length at least three; (*ii*) a *P-node* (parallel composition), if skel(ν) is a bundle of at least three parallel edges; (*iii*) an *R-node* (rigid composition), if skel(ν) is a triconnected graph. A degree-1 node of T is a *Q-node* and represents a single edge of G. A *real edge* (resp. *virtual edge*) in skel(ν) corresponds to a Q-node (resp., to an S-, P-, or R-node) adjacent to ν in T. Let e be a designated edge of G, called the *reference edge* of G, let ρ be the Q-node of T corresponding to e, and let T be rooted at ρ. For any P-, S-, or R-node ν of T distinct from the root child, skel(ν) has a virtual edge, called *reference edge* of skel(ν) and of ν, associated with a virtual edge in the skeleton of its parent. The reference edge of the root child of T is the edge corresponding to ρ. For every node $\nu \neq \rho$, the *pertinent graph* G_ν *of* ν is the subgraph of G whose edges correspond to the Q-nodes in the subtree of T rooted at ν. We also say that G_ν is a *component of* G. The pertinent graph G_ρ of the root ρ coincides with the reference edge of G. If H is a rectilinear representation or an RU rectilinear representation of G, its restriction H_ν to G_ν is a *rectilinear component* or an *RU rectilinear component* of H. As in [7, 11, 15, 16, 18, 20, 23], we implicitly assume to work with a *normalized* SPQR-tree, in which every S-node has exactly two children. Every SPQR-tree can be normalized in $O(n)$ time by recursively splitting an S-node with more than two children into multiple S-nodes with two children. If G has n vertices, a normalized SPQR-tree of G still has $O(n)$ nodes.

RU-Spirality. Let G be a biconnected planar digraph and consider an SPQR-tree T of G rooted at a Q-node ρ, corresponding to a reference edge (s, t). Assume for convenience that the vertices of G are labeled with an st-numbering [24] of G (see, e.g., Figure 8a). Let H be an orthogonal representation of G with the reference edge $G_\rho = (s, t)$ in the external



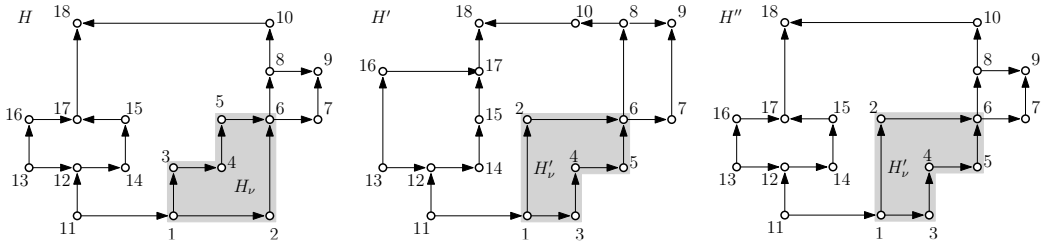
(a) G **(b)** H **(c)** H'

Figure 5 Illustration of the concept of RU-spirality. The two representations in (b) and (c) have the same value of σ_ν but different RU spiralities.

face, let H_ν be a component of H (i.e., the restriction of H to G_ν), and let $\{u, v\}$ be the poles of ν, where u precedes v in the st-numbering. We say that u and v are the *first-pole* and the *second-pole* of ν, respectively. Note that we are not assuming any relationship between the st-numbering and the orientation of the edges of G. For each pole $w \in \{u, v\}$, let $\text{intdeg}_\nu(w)$ and $\text{extdeg}_\nu(w)$ be the degree of w inside and outside H_ν, respectively. We define two (possibly coincident) *alias vertices* of w, denoted by w' and w'', as follows: (i) if $\text{intdeg}_\nu(w) = 1$, then $w' = w'' = w$; (ii) if $\text{intdeg}_\nu(w) = \text{extdeg}_\nu(w) = 2$, then w' and w'' are dummy vertices, each splitting one of the two distinct edge segments incident to w outside H_ν; (iii) if $\text{intdeg}_\nu(w) > 1$ and $\text{extdeg}_\nu(w) = 1$, then $w' = w''$ is a dummy vertex that splits the edge segment incident to w outside H_ν.

Let A^w be the set of distinct alias vertices of a pole w. Let P^{uv} be any simple undirected path from u to v inside H_ν and let $u' \in A^u$ and $v' \in A^v$ be two alias vertices of u and of v, respectively. The path $S^{u'v'}$ obtained concatenating (u', u), P^{uv}, and (v, v') is a *spine* of H_ν. Denote by $n(S^{u'v'})$ the number of right turns minus the number of left turns encountered along $S^{u'v'}$ moving from u' to v'. The *rectilinear spirality* $\sigma(H_\nu)$ of H_ν is either an integer or a semi-integer number, defined based on the following cases: (i) If $A^u = \{u'\}$ and $A^v = \{v'\}$ then $\sigma(H_\nu) = n(S^{u'v'})$. (ii) If $A^u = \{u'\}$ and $A^v = \{v', v''\}$ then $\sigma(H_\nu) = \frac{n(S^{u'v'}) + n(S^{u'v''})}{2}$. (iii) If $A^u = \{u', u''\}$ and $A^v = \{v'\}$ then $\sigma(H_\nu) = \frac{n(S^{u'v'}) + n(S^{u''v'})}{2}$. (iv) If $A^u = \{u', u''\}$ and $A^v = \{v', v''\}$ assume, without loss of generality, that (u, u') succeeds (u, u'') clockwise around u and that (v, v') precedes (v, v'') clockwise around v; then $\sigma(H_\nu) = \frac{n(S^{u'v'}) + n(S^{u''v''})}{2}$.

For brevity, in the following we often denote by σ_ν the rectilinear spirality of an RU representation of G_ν. Let $\{S, N, W, E\}$ denote the set of the four possible sides (North, South, East, West) by which an edge can be incident to a vertex in an RU representation. The *rectilinear-upward spirality* (*RU-spirality* for short) of H_ν, denoted by $\tau(H_\nu)$ (or simply by τ_ν), is a tuple $\langle \sigma_\nu, \varphi_u, \varphi_v \rangle$, where σ_ν is the rectilinear spirality of H_ν and where $\varphi_w = (S_w, N_w, W_w, E_w)$ specifies the arrangement of the internal and external edges of H_ν incident to a pole $w \in \{u, v\}$, with respect to the four sides S (South), N (North), W (West), and E (East). Precisely, for each $D \in \{S, N, W, E\}$ and $w \in \{u, v\}$, we have $D_w \in \{\mathsf{free}, \mathsf{int}, \mathsf{ext}\}$ in such a way that: $D_w = \mathsf{free}$ if no edge is incident to w from side D; $D_w = \mathsf{int}$ if there is an edge of H_ν (i.e., an edge internal to H_ν) incident to w from side D; $D_w = \mathsf{ext}$ if there is an edge of H not in H_ν (i.e., an edge external to H_ν) incident to w from side D. We call σ_ν the *rectilinear spirality* of τ_ν; φ_u and φ_v the *pole side specifications* of τ_ν. Figure 5 shows an illustration of the concept of RU-spirality. In Figure 5a there is a digraph G with a highlighted S-component G_ν with first-pole u and second-pole v.



(a) G **(b)** H **(c)** H'

Figure 6 Illustration of the concept of substitution. The RU representation H'' in (c) is obtained by substituting H_ν with H'_ν in H. The first-pole of ν is vertex 1 and the second-pole of ν is vertex 6. We have $\tau(H_\nu) = \tau(H'_\nu) = \langle -\frac{1}{2}, (\mathsf{free}, \mathsf{int}, \mathsf{ext}, \mathsf{int}), (\mathsf{int}, \mathsf{ext}, \mathsf{int}, \mathsf{ext})\rangle$.

Figure 5b and Figure 5c show two different RU representations H and H' of G. In H we have $\tau_\nu = \langle 3, \varphi_u = (\mathsf{free}, \mathsf{ext}, \mathsf{int}, \mathsf{ext}), \varphi_v = (\mathsf{ext}, \mathsf{int}, \mathsf{ext}, \mathsf{free})\rangle$ and in H' we have $\tau_\nu = \langle 3, \varphi_u = (\mathsf{ext}, \mathsf{int}, \mathsf{free}, \mathsf{ext}), \varphi_v = (\mathsf{free}, \mathsf{ext}, \mathsf{ext}, \mathsf{int})\rangle$.

Note that, denoted by G' the subgraph of G consisting of G_ν plus the external edges incident to the poles of ν, the RU-spirality for an RU representation H_ν of G_ν can also be defined referring to an RU representation of G' that contains H_ν, rather than to an RU representation of G. If we are able to construct an RU representation H' of G' such that its restriction H_ν to G_ν has RU-spirality τ_ν, then we say that G_ν (or simply ν) *admits* RU-spirality τ_ν. Observe that, even if H_ν admits RU-spirality τ_ν, it might not exist an RU representation of G whose restriction to G_ν has spirality τ_ν.

Substituting components with the same RU-spirality. We extend the results in [11, 18] to show that components with the same RU spirality are "interchangeable". Let H and H' be two different RU representations of G with the same reference edge G_ρ on the external face. Also let H_ν and H'_ν be the restrictions of H and H' to the same component G_ν. If $\tau(H_\nu) = \tau(H'_\nu)$, the operation $\mathrm{Sub}(H_\nu, H'_\nu)$ of *substituting* H_ν with H'_ν in H defines a new plane digraph H'' with an angle labeling such that the restriction of H'' to G_ν coincides with H'_ν, while the restriction of H'' to $G \setminus G_\nu$ stays as in H. More formally, let u and v be the first-pole and second-pole of ν, respectively. The external boundary of H_ν contains a left path p_l and a right path p_r, such that p_l (resp. p_r) goes from u to v traversing the external boundary of H_ν clockwise (resp. counterclockwise). Let f_l and f_r be the faces of H outside H_ν and incident to p_l and p_r, respectively. With respect to H'_ν and H', define p'_l, p'_r, f'_l, f'_r analogously. Since $\tau(H_\nu) = \tau(H'_\nu)$, the circular sequence of angles at each pole $w \in \{u, v\}$ is the same in H and in H', namely the angles at w internal and external to G_ν are the same in H and H'. The digraph H'' is defined as follows:

- H'' has the same set of vertices and edges as G.
- The planar embedding of H'' is such that: all the faces of H outside H_ν and distinct from f_l and f_r, as well as all faces of H'_ν, are also faces of H''. Further, H'' has two faces f''_l and f''_r obtained by replacing p_l with p'_l and p_r with p'_r in the boundary of f_l and of f_r, respectively.
- The angle labeling of H'' is such that: (i) all the angles at the vertices of G not belonging to G_ν are those in H; (ii) all the angles at the vertices of G_ν distinct from u and v are those in H'_ν; (iii) for each pole $w \in \{u, v\}$, the internal and external angles at w are defined as in H or, equivalently, as in H' (they are the same as $\tau(H_\nu) = \tau(H'_\nu)$).

The following result proves that H'' is an RU representation.

▶ **Theorem 5.** *Let G be a biconnected planar digraph, T be an SPQR-tree of G with respect to a given reference e, and ν be a non-root node of T. Let H and H' be two different RU representations of G with e on the external face, and let H_ν and H'_ν be the restrictions of H and of H' to G_ν, respectively. If $\tau(H_\nu) = \tau(H'_\nu)$ then the graph H'' defined by $\mathrm{Sub}(H_\nu, H'_\nu)$ is an RU representation of G.*

Proof. The fact that the planar embedding and the labeling of H'' describe a rectilinear planar representation of G is proved in [11, 18], as a consequence that H_ν and H'_ν have the same rectilinear spirality. We now orient H'' in such a way that, for an arbitrarily chosen edge $e = (x, y)$ of H'_ν, the vertices x and y have the same relative positions as in H'_ν. This implies that for each edge e' of G_ν, the relative position of the end-vertices of e' in H'' remains the same as in H'. Also, for each side $\{S, N, W, E\}$ of w, either this side is free in both H and H', or it is occupied either by an edge internal to G_ν or by an edge external to G_ν in both H and H'. This implies that, with the chosen orientation, for each edge e'' of $G \setminus G_\nu$ the relative position of the end-vertices of e'' in H'' is the same as in H. It follows that, with the chosen orientation, no edge of H'' is downward. ◀

Based on Theorem 5, in order to test RU planarity of a biconnected digraph G with a given reference edge on the external face, we exploit a dynamic programming technique that visits a rooted SPQR-tree T of G bottom-up. At each visited node ν of T, and for each RU spirality τ_ν admitted by ν, we store at ν a pair $\langle \tau_\nu, H_\nu \rangle$, where H_ν is just one RU representation of G_ν with spirality τ_ν, called a *representative of* τ_ν. The set of all pairs $\langle \tau_\nu, H_\nu \rangle$ is the *feasible set of* ν and is denoted by Σ_ν. Observe that, if G_ν has n_ν vertices and if $\tau_\nu \in \Sigma_\nu$, the rectilinear spirality σ_ν in τ_ν cannot exceed n_ν, as we can make at most n_ν right or n_ν left turns. Also, for each value σ_ν, the number of RU spiralities τ_ν in Σ_ν with rectilinear spirality σ_ν is bounded by a constant. Hence, we have the following.

▶ **Property 2.** *For any component G_ν with n_ν vertices, $|\Sigma_\nu| = O(n_\nu)$. Also, for each $\tau_\nu \in \Sigma_\nu$, the corresponding rectilinear spirality σ_ν belongs to the interval $[-n_\nu, n_\nu]$.*

5.2 Testing Series-Parallel Digraphs in Polynomial Time

When the SPQR-tree T of a biconnected graph G does not have R-nodes, G is a *series-parallel graph*, or simply an *SP-graph*. Also, T is called the SPQ-tree of G. In this section we assume that G is an *SP-digraph*, i.e., a digraph whose underlying undirected graph is an SP-graph. We also assume that T is normalized. We prove the following lemmas.

▶ **Lemma 6.** *Let ν be a Q-node of T. We can compute Σ_ν in $O(1)$ time.*

Proof. G_ν is a directed edge $e = (u, v)$ of G. In any RU representation of G, edge e is either leftward, or rightward, or upward. For each of these three possibilities, we have to consider the $O(1)$ possible arrangements of the edges incident to e on the different sides of u and v, each of them defining a different spirality τ_ν. Thus Σ_ν is constructed in $O(1)$ time. ◀

▶ **Lemma 7.** *Let ν be an S-node of T with children μ_1 and μ_2, and let n_1^ν and n_2^ν be the number of nodes in G_{μ_1} and G_{μ_2}, respectively. If Σ_{μ_1} and Σ_{μ_2} are given, then we can compute Σ_ν in $O(n_1^\nu \cdot n_2^\nu)$ time.*

Sketch of Proof. For each pair $\tau_{\mu_1} \in \Sigma_{\mu_1}$ and $\tau_{\mu_2} \in \Sigma_{\mu_2}$, let H_{μ_1} and H_{μ_2} be the representatives of τ_{μ_1} and τ_{μ_2}, respectively. Let u_i and v_i be the first-and second-pole of μ_i, respectively, with $i = 1, 2$. Clearly $v_1 = u_2$. Suppose that for each side $D \in \{S, N, W, E\}$

one of these three cases holds: (*i*) $D_{v_1} = \mathsf{free}$ in τ_{μ_1} and $D_{u_2} = \mathsf{free}$ in τ_{μ_2}; (*ii*) $D_{v_1} = \mathsf{int}$ in τ_{μ_1} and $D_{u_2} = \mathsf{ext}$ in τ_{μ_2}; (*iii*) $D_{v_1} = \mathsf{ext}$ in τ_{μ_1} and $D_{u_2} = \mathsf{int}$ in τ_{μ_2}. If so the pole side specifications of τ_{μ_1} and τ_{μ_2} are compatible, and we can construct an RU representation H_ν by gluing together H_{μ_1} and H_{μ_2} at the common pole $v_1 = u_2$; the rectilinear spirality σ_ν in τ_ν is computed based on σ_{μ_1}, σ_{μ_2}, and on the pole side specifications in τ_{μ_1} and τ_{μ_2}. ◀

The next structural lemma, which is proven by induction on the depth of a normalized rooted SPQ-tree T, is given in [17]. Corollary 9 follows by combining Lemma 7 and Lemma 8.

▶ **Lemma 8** ([17]). *Let T be a normalized rooted SPQ-tree of an n-vertex SP-digraph G, and let $\mathcal{S}$ be the set of all S-nodes of T. We have $\sum_{\nu \in \mathcal{S}} n_1^\nu \cdot n_2^\nu = O(n^2)$, where n_1^ν and n_2^ν are the number of vertices in the pertinent graphs of the two children of ν.*

▶ **Corollary 9.** *Let T be a normalized rooted SPQ-tree of an n-vertex SP-digraph G. Assume that T is visited bottom-up and that when we visit a node the feasible sets of its children are known. Then, the feasible sets of all S-nodes of T can be computed in overall $O(n^2)$ time.*

The next lemma is about the feasible sets of P-nodes. Theorem 11 summarizes the main result of this subsection.

▶ **Lemma 10.** *Let ν be a P-node of T with children $\mu_1, \mu_2, \dots \mu_h$ $(h = 2, 3)$. If Σ_{μ_1} and Σ_{μ_2} are given, then we can compute Σ_ν in $O(n)$ time.*

Proof. Let u and v be the first-pole and the second-pole of v, respectively. By definition of P-node, u and v are also the first-pole and the second-pole of each child of ν. Denote by n_ν the number of vertices of G_ν. Suppose first that ν is a P-node with three children μ_1, μ_2, and μ_3. Any planar embedding of G defines a planar embedding of skel(ν). If for simplicity we topologically imagine v above u, in any given embedding of skel(ν) the three children of ν (namely, the edges of skel(ν) that correspond to these children) occur from left to right in some order (equivalently, this order coincides with the circular order in which these children are encountered around v moving counterclockwise from the reference edge of skel(ν)). Suppose that for a given embedding ϕ of skel(ν), we rename the three children of ν as μ_l, μ_c, and μ_r, if they occur in this left-to-right order in ϕ. Let H be any rectilinear representation of G that induces for skel(ν) the embedding ϕ. Also denote by σ_ν, σ_{μ_l}, σ_{μ_c}, and σ_{μ_r} the rectilinear spirality values of the restrictions of H to G_ν, G_{μ_l}, G_{μ_c}, and G_{μ_r}, respectively. It is proved in [11] that the following relationship holds: $\sigma_\nu = \sigma_{\mu_l} - 2 = \sigma_{\mu_c} = \sigma_{\mu_r} + 2$. Clearly, since an RU representation is in particular a rectilinear representation, then the same relationship must be verified for any RU representation that induces the embedding ϕ for skel(ν). Hence, as done in [11], for each candidate rectilinear spirality value $\sigma_\nu \in [-n_\nu, n_\nu]$ (see Property 2) and for each possible embedding ϕ of skel(ν), one can check in $O(1)$ time whether there exist three elements $\tau_{\mu_l} \in \Sigma_{\mu_l}$, $\tau_{\mu_c} \in \Sigma_{\mu_c}$, and $\tau_{\mu_r} \in \Sigma_{\mu_r}$ such that the corresponding rectilinear spiralities σ_{μ_l}, σ_{μ_c}, and σ_{μ_r} satisfy the above relationship. If not, then the target rectilinear spirality value σ_ν is not feasible, otherwise suppose that such elements τ_{μ_l}, τ_{μ_c}, and τ_{μ_r} exist. To check whether we can combine them into an RU spirality τ_ν having rectilinear spirality σ_ν, we have to test the compatibility of the pole side specifications for each pole $w \in \{u, v\}$. This compatibility can be checked with the following simple considerations. Since ν has three children, w has degree four in G. Also, each of the three components G_{μ_l}, G_{μ_c}, and G_{μ_r} contains exactly one edge incident to w, while the fourth edge incident to w is external to all the three components. Hence, to have compatibility, we must have that in the pole specification of each τ_{μ_j} $(j = l, c, r)$ there is exactly one side of w with value int and each other side of w with value ext. In particular, call D_w the side of w in the pole specification

of τ_{μ_l} for which $D_w = \mathsf{int}$. To fix the ideas, assume that $D_w = W_w$, i.e., the edge of G_{μ_l} incident w occupies the West side of w (the cases $D_w = S_w$, $D_w = N_w$, and $D_w = E_w$ are treated in a similar way). This means that $\varphi_w = (\mathsf{ext}, \mathsf{ext}, \mathsf{int}, \mathsf{ext})$ in τ_{μ_l}. Thus, in order to have compatibility at w it must be $\varphi_w = (\mathsf{int}, \mathsf{ext}, \mathsf{ext}, \mathsf{ext})$ in τ_{μ_c} and $\varphi_w = (\mathsf{ext}, \mathsf{ext}, \mathsf{ext}, \mathsf{int})$ in τ_{μ_r}. If there is compatibility at the pole u and at the pole v, we can glue together the representatives H_{μ_l}, H_{μ_c}, H_{μ_r} into an RU representation H_ν with rectilinear spirality σ_ν, and we insert $\tau_\nu = \langle \sigma_\nu, H_\nu \rangle$ in Σ_ν; otherwise we discard the triplet $\tau_{\mu_l}, \tau_{\mu_c}, \tau_{\mu_r}$.

The described procedure for constructing Σ_ν takes $O(n)$ time because: (i) by Property 2 there are $O(n)$ possible target rectilinear spirality values to consider; (ii) for each target spirality value, skel(ν) has 6 distinct planar embeddings to consider; (iii) for each embedding of skel(ν) we can check in $O(1)$ time which triplets $\tau_{\mu_l}, \tau_{\mu_c}, \tau_{\mu_r}$ that satisfy the relation $\sigma_\nu = \sigma_{\mu_l} - 2 = \sigma_{\mu_c} = \sigma_{\mu_r} + 2$ (see also [11]), and for each of these triplets we can also check in $O(1)$ time the compatibility of the pole side specifications.

If ν is a P-node with two children, the strategy for constructing Σ_ν is exactly the same. However in this case, skel(ν) has only two embeddings to consider for each target value of rectilinear spirality σ_ν. Also, for each of these two embeddings, the relationship between σ_ν and the rectilinear spiralities of the children of ν, as well as the compatibility conditions for the pole side specifications, may require to analyze more cases, whose number is however still bounded by a constant (see [11] for details about the relationships between a rectilinear representation of ν and those of its two children). ◀

▶ **Theorem 11.** *Let G be an n-vertex SP-digraph. There exists an $O(n^3)$-time algorithm that tests whether G admits an RU representation, and that computes one in the affirmative case.*

Sketch of Proof. Let T be the SPQ-tree of G. For each Q-node ρ of T, the algorithm considers T rooted at ρ and performs a post-order visit of T to tests whether G admits an RU representation with the reference edge G_ρ on the external face. It first computes Σ_ν for each leaf ν of T, that is, for each Q-node of T distinct from ρ. Then, for each internal node ν of T distinct from ρ the algorithm computes Σ_ν by using the feasible sets of the children of ν, by means of Lemma 7 or of Lemma 10 depending on whether ν is an S-node or a P-node. If Σ_ν is empty then G does not have an RU representation with G_ρ on the external face, and the algorithm starts visiting T rooted at another Q-node. Suppose vice versa that the algorithm achieves the root child ν and that Σ_ν is not empty. The algorithm checks if there is $\tau_\nu \in \Sigma_\nu$ whose H_ν can be glued together with a straight-line representation of the reference edge G_ρ, which is oriented either upward, or leftward, or rightward.

Regarding the time complexity, the algorithm has to test $O(n)$ rooted SPQ-trees. For each tree, the feasible sets of all Q-nodes can be computed in overall $O(n)$ time by Lemma 6, those of all S-nodes can be computed in $O(n^2)$ time by Corollary 9, and those of all P-nodes can be computed in $O(n^2)$ time by Lemma 10. Finally, the condition at the root can be checked in $O(n)$ time. Hence, the whole algorithm can be executed in $O(n^3)$ time. ◀

5.3 FPT Testing Algorithm by the Number of Sources and Sinks

Let G be a biconnected digraph. A vertex of G that is either a source or a sink of G is called a *switch* [10]. We sketch the description of an FPT algorithm for Rectilinear-Upward Planarity Testing parameterized by the number k of switches of G.

Let T be a rooted SPQR-tree of G and let ν be any node of T. It can be shown that: (*i*) if G_ν does not contain any switches of G, then it can only admit a constant number of rectilinear spirality values, and hence a constant number of RU spiralities; (*ii*) otherwise, the possible values of rectilinear spirality admitted by G_ν is a linear function of k.

The FPT algorithm extends the dynamic programming of Section 5.2 so to handle R-nodes. When an R-node ν of a rooted SPQR-tree is visited, we consider the two possible planar embeddings of its skeleton, and for each of these two embeddings we consider every possible upward planar embedding and every possible target RU spirality τ_ν; then, we test whether G_ν admits τ_ν. If so, as for S-nodes and P-nodes, we construct an RU representation H_ν and we insert $\langle \tau_\nu, H_\nu \rangle$ in Σ_ν; otherwise, τ_ν is discarded. To perform the test for each target value τ_ν, we partition the children of ν into two sets A and B. Namely, a child μ of ν is inserted in A if G_μ contains at least one switch of G, otherwise we insert μ in B. Clearly $|A| = O(k)$, while for each element in B the size of the feasible set is constant. Then, for each combination of fixed RU spiralities in the feasible sets of the elements in A, we solve a constrained RU planarity testing problem that: (a) forces G_ν to have the target rectilinear spirality σ_ν (associated with τ_ν); (b) preserves the chosen combination of RU spiralities for the elements of A; and (c) guarantees that the pertinent graphs of the nodes in B have one of the constantly many RU spiralities in their feasible sets. We prove that this test can be executed in $O(n)$ time by using the 2-SAT model of Section 4, enriched with $O(n)$ number of constraints. Since there are $O(k^k) = 2^{O(k \log k)}$ combinations of RU spiralities for the elements in A, and since there are $O(4^k) = O(2^{2k})$ upward planar embeddings for each of the two possible planar embeddings of an R-node, we get the following.

▶ **Lemma 12.** *Let ν be an R-node of T and let $\mu_1, \ldots, \mu_h$ be its children. Given the feasible set Σ_{μ_i} for each $i \in \{1, \ldots, h\}$, we can compute Σ_ν in $2^{O(k \cdot \log k + 2k)} \cdot O(n)$ time.*

By Lemma 12, for processing all R-nodes of T we spend in total $2^{O(k \cdot \log k + 2k)} \cdot O(n^2)$. For an S-node or a P-node ν, we use exactly the same strategy as in Section 5.2. However, since the sizes of the feasible sets of all children of ν, and of ν itself, are now $O(k)$, Σ_ν can be constructed in $O(k^2)$ time if ν is an S-node and in time $O(k)$ if ν is a P-node; hence we spend $O(nk^2)$ for processing all S- and P-nodes of T. Finally, since every RU representation of G has at least one source and one sink in its external face, it suffices to test $O(k)$ possible rooted SPQR-trees, thus saving the extra $O(n)$ factor of Theorem 11. The following holds.

▶ **Theorem 13.** *Let G be a planar digraph with k switches. There exists an $2^{O(k \cdot \log k + 2k)} \cdot O(n^2)$-time algorithm that tests whether G is rectilinear-upward planar and that computes an RU representation of G in the positive case.*

A byproduct of the previous theorem is the following corollary for the well-known family of *st*-digraphs, i.e., digraphs with a single source and a single sink.

▶ **Corollary 14.** *The Rectilinear-Upward Planarity Testing problem can be solved in $O(n^2)$ time for planar st-digraphs with n vertices.*

6 Open Problems

The NP-hardness of Rectilinear-Upward Planarity Testing holds when the embedding can vary while the linear-time solution holds for upward plane digraphs. Also the testing is trivial if a rectilinear embedding is given. Establishing the complexity of the problem when a planar embedding (neither rectilinear nor upward) is fixed remains an open question. Moreover, our results in the variable embedding setting consider biconnected graphs; extending these results to simply connected instances is a topic for future exploration. Lastly, there is potential for future research in improving the time complexity for series-parallel digraphs.

References

1 Sarmad Abbasi, Patrick Healy, and Aimal Rextin. Improving the running time of embedded upward planarity testing. *Inf. Process. Lett.*, 110(7):274–278, 2010. doi:10.1016/j.ipl.2010.02.004.

2 Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theor. Comput. Sci.*, 61:175–198, 1988. doi:10.1016/0304-3975(88)90123-5.

3 Paola Bertolazzi, Giuseppe Di Battista, Giuseppe Liotta, and Carlo Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, 12(6):476–497, 1994. doi:10.1007/BF01188716.

4 Paola Bertolazzi, Giuseppe Di Battista, Carlo Mannino, and Roberto Tamassia. Optimal upward planarity testing of single-source digraphs. *SIAM J. Comput.*, 27(1):132–169, 1998. doi:10.1137/S0097539794279626.

5 Carla Binucci, Walter Didimo, and Maurizio Patrignani. Upward and quasi-upward planarity testing of embedded mixed graphs. *Theor. Comput. Sci.*, 526:75–89, 2014. doi:10.1016/j.tcs.2014.01.015.

6 Franz-Josef Brandenburg, David Eppstein, Michael T. Goodrich, Stephen G. Kobourov, Giuseppe Liotta, and Petra Mutzel. Selected open problems in graph drawing. In Giuseppe Liotta, editor, *Graph Drawing, 11th International Symposium, GD 2003, Perugia, Italy, September 21-24, 2003, Proceedings*, volume 2912 of *Lecture Notes in Computer Science*, pages 515–539. Springer, 2003. doi:10.1007/978-3-540-24595-7_55.

7 Steven Chaplick, Emilio Di Giacomo, Fabrizio Frati, Robert Ganian, Chrysanthi N. Raftopoulou, and Kirill Simonov. Parameterized algorithms for upward planarity. In Xavier Goaoc and Michael Kerber, editors, *38th International Symposium on Computational Geometry, SoCG 2022, June 7-10, 2022, Berlin, Germany*, volume 224 of *LIPIcs*, pages 26:1–26:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SoCG.2022.26.

8 Steven Chaplick, Emilio Di Giacomo, Fabrizio Frati, Robert Ganian, Chrysanthi N. Raftopoulou, and Kirill Simonov. Testing upward planarity of partial 2-trees. In Patrizio Angelini and Reinhard von Hanxleden, editors, *Graph Drawing and Network Visualization – 30th International Symposium, GD 2022, Tokyo, Japan, September 13-16, 2022, Proceedings*, volume 13764 of *Lecture Notes in Computer Science*, pages 175–187. Springer, 2022. doi:10.1007/978-3-031-22203-0_13.

9 Sabine Cornelsen and Andreas Karrenbauer. Accelerated bend minimization. *J. Graph Algorithms Appl.*, 16(3):635–650, 2012. doi:10.7155/jgaa.00265.

10 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

11 Giuseppe Di Battista, Giuseppe Liotta, and Francesco Vargiu. Spirality and optimal orthogonal drawings. *SIAM J. Comput.*, 27(6):1764–1811, 1998. doi:10.1137/S0097539794262847.

12 Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Sketched representations and orthogonal planarity of bounded treewidth graphs. In Daniel Archambault and Csaba D. Tóth, editors, *Graph Drawing and Network Visualization – 27th International Symposium, GD 2019, Prague, Czech Republic, September 17-20, 2019, Proceedings*, volume 11904 of *Lecture Notes in Computer Science*, pages 379–392. Springer, 2019. doi:10.1007/978-3-030-35802-0_29.

13 Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Orthogonal planarity testing of bounded treewidth graphs. *J. Comput. Syst. Sci.*, 125:129–148, 2022. doi:10.1016/j.jcss.2021.11.004.

14 Walter Didimo, Emilio Di Giacomo, Giuseppe Liotta, Fabrizio Montecchiani, and Giacomo Ortali. On the parameterized complexity of bend-minimum orthogonal planarity. *CoRR*, abs/2308.13665, 2023. doi:10.48550/arXiv.2308.13665.

15 Walter Didimo, Francesco Giordano, and Giuseppe Liotta. Upward spirality and upward planarity testing. *SIAM J. Discret. Math.*, 23(4):1842–1899, 2009. doi:10.1137/070696854.

16 Walter Didimo, Michael Kaufmann, Giuseppe Liotta, and Giacomo Ortali. Rectilinear planarity of partial 2-trees. In Patrizio Angelini and Reinhard von Hanxleden, editors, *Graph Drawing and Network Visualization – 30th International Symposium, GD 2022, Tokyo, Japan, September 13-16, 2022, Proceedings*, volume 13764 of *Lecture Notes in Computer Science*, pages 157–172. Springer, 2022. doi:10.1007/978-3-031-22203-0_12.

17 Walter Didimo, Michael Kaufmann, Giuseppe Liotta, and Giacomo Ortali. Rectilinear planarity of partial 2-trees. *CoRR*, abs/2208.12558, 2022. doi:10.48550/arXiv.2208.12558.

18 Walter Didimo, Michael Kaufmann, Giuseppe Liotta, and Giacomo Ortali. Computing bend-minimum orthogonal drawings of plane series–parallel graphs in linear time. *Algorithmica*, 2023. doi:10.1007/s00453-023-01110-6.

19 Walter Didimo, Michael Kaufmann, Giuseppe Liotta, and Giacomo Ortali. Rectilinear planarity of partial 2-trees. *J. of Graph Algorithms Appl., special issue on GD 2022.*, 2023. to appear.

20 Walter Didimo and Giuseppe Liotta. Computing orthogonal drawings in a variable embedding setting. In Kyung-Yong Chwa and Oscar H. Ibarra, editors, *Algorithms and Computation, 9th International Symposium, ISAAC '98, Taejon, Korea, December 14-16, 1998, Proceedings*, volume 1533 of *Lecture Notes in Computer Science*, pages 79–88. Springer, 1998. doi:10.1007/3-540-49381-6_10.

21 Walter Didimo and Giuseppe Liotta. *Graph Visualization and Data Mining*, chapter 3, pages 35–63. John Wiley & Sons, Ltd, 2006. doi:10.1002/9780470073049.ch3.

22 Walter Didimo, Giuseppe Liotta, Giacomo Ortali, and Maurizio Patrignani. Optimal orthogonal drawings of planar 3-graphs in linear time. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 806–825. SIAM, 2020. doi:10.1137/1.9781611975994.49.

23 Walter Didimo, Giuseppe Liotta, and Maurizio Patrignani. HV-planarity: Algorithms and complexity. *J. Comput. Syst. Sci.*, 99:72–90, 2019. doi:10.1016/j.jcss.2018.08.003.

24 Shimon Even and Robert Endre Tarjan. Corrigendum: Computing an *st*-numbering. TCS 2(1976):339-344. *Theor. Comput. Sci.*, 4(1):123, 1977.

25 Uli Fossmeier and Michael Kaufmann. An approach to bend-minimal upward drawing. In *Graph Drawing, 1th International Symposium, GD 1993, Paris, France*, pages 27–29, 1993.

26 Ulrich Fößmeier and Michael Kaufmann. On bend-minimum orthogonal upward drawing of directed planar graphs. In Roberto Tamassia and Ioannis G. Tollis, editors, *Graph Drawing, DIMACS International Workshop, GD '94, Princeton, New Jersey, USA, October 10-12, 1994, Proceedings*, volume 894 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 1994. doi:10.1007/3-540-58950-3_356.

27 Fabrizio Frati. Planar rectilinear drawings of outerplanar graphs in linear time. *Comput. Geom.*, 103:101854, 2022. doi:10.1016/j.comgeo.2021.101854.

28 Ashim Garg and Roberto Tamassia. A new minimum cost flow algorithm with applications to graph drawing. In Stephen C. North, editor, *Graph Drawing, Symposium on Graph Drawing, GD '96, Berkeley, California, USA, September 18-20, Proceedings*, volume 1190 of *Lecture Notes in Computer Science*, pages 201–216. Springer, 1996. doi:10.1007/3-540-62495-3_49.

29 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001. doi:10.1137/S0097539794277123.

30 Carsten Gutwenger and Petra Mutzel. A linear time implementation of spqr-trees. In Joe Marks, editor, *Graph Drawing, 8th International Symposium, GD 2000, Colonial Williamsburg, VA, USA, September 20-23, 2000, Proceedings*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 2000. doi:10.1007/3-540-44541-2_8.

31 Md. Manzurul Hasan and Md. Saidur Rahman. No-bend orthogonal drawings and no-bend orthogonally convex drawings of planar graphs (extended abstract). In Ding-Zhu Du, Zhenhua Duan, and Cong Tian, editors, *Computing and Combinatorics – 25th International Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, Proceedings*, volume 11653 of *Lecture Notes in Computer Science*, pages 254–265. Springer, 2019. doi:10.1007/978-3-030-26176-4_21.

32 Patrick Healy and Karol Lynch. Two fixed-parameter tractable algorithms for testing upward planarity. *Int. J. Found. Comput. Sci.*, 17(5):1095–1114, 2006. doi:10.1142/S0129054106004285.

33 John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973. doi:10.1137/0202012.

34 Michael D. Hutton and Anna Lubiw. Upward planning of single-source acyclic digraphs. *SIAM J. Comput.*, 25(2):291–311, 1996. doi:10.1137/S0097539792235906.

35 Bart M. P. Jansen, Liana Khazaliya, Philipp Kindermann, Giuseppe Liotta, Fabrizio Montecchiani, and Kirill Simonov. Upward and orthogonal planarity are W[1]-hard parameterized by treewidth. *CoRR*, abs/2309.01264, 2023. doi:10.48550/arXiv.2309.01264.

36 Michael Jünger and Petra Mutzel, editors. *Graph Drawing Software*. Springer, 2004. doi:10.1007/978-3-642-18638-7.

37 M. R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967. doi:10.1002/malq.19670130104.

38 Takao Nishizeki and Md. Saidur Rahman. *Planar Graph Drawing*, volume 12 of *Lecture Notes Series on Computing*. World Scientific, 2004. doi:10.1142/5648.

39 Maurizio Patrignani. Planarity testing and embedding. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 1–42. Chapman and Hall/CRC, 2013.

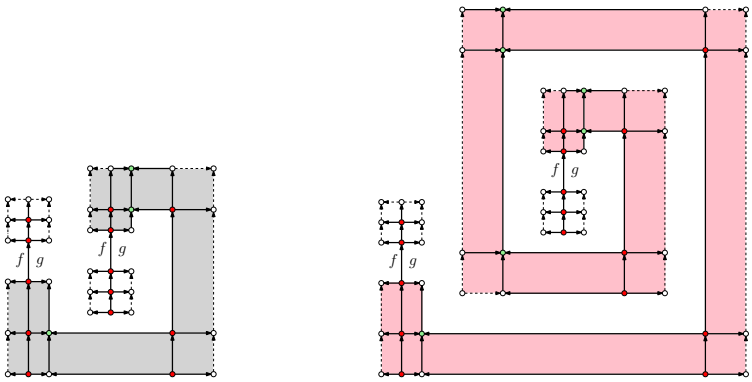
40 Md. Saidur Rahman, Noritsugu Egi, and Takao Nishizeki. No-bend orthogonal drawings of subdivisions of planar triconnected cubic graphs. *IEICE Trans. Inf. Syst.*, 88-D(1):23–30, 2005. URL: http://search.ieice.org/bin/summary.php?id=e88-d_1_23&category=D&year=2005&lang=E&abst=.

41 Md. Saidur Rahman, Takao Nishizeki, and Mahmuda Naznin. Orthogonal drawings of plane graphs without bends. *J. Graph Algorithms Appl.*, 7(4):335–362, 2003. doi:10.7155/jgaa.00074.

42 Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. doi:10.1137/0216030.

43 Roberto Tamassia and Giuseppe Liotta. Graph drawing. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, pages 1163–1185. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035315.ch52.

A Appendix



(a) Tentril T_1. **(b)** Tendril T_2.

Figure 7 Tendrils T_1 (a) and T_2 (b). Red vertices have three outgoing edges. Green vertices have three incoming edges. Solid edges are incident either to a red or to a green vertex (or both).

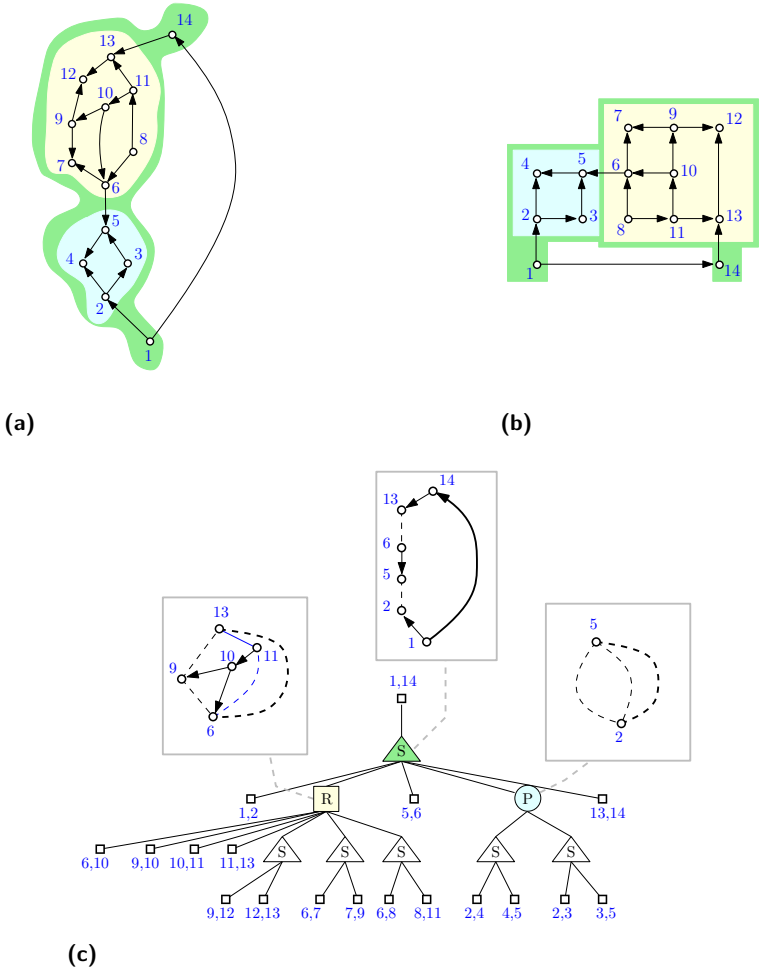


Figure 8 (a) A digraph G with three highlighted components (an S-, an R- and a P-component); (b) an RU representation of G. (c) The SPQR-tree of G with reference edge $(1, 14)$; the skeletons of the highlighted components are shown: dashed edges are virtual and the reference edge is thicker. Q-nodes are labeled with the end-vertices of their corresponding edges.

A Unified Worst Case for Classical Simplex and Policy Iteration Pivot Rules

Yann Disser
TU Darmstadt, Germany

Nils Mosis
TU Darmstadt, Germany

Abstract

We construct a family of Markov decision processes for which the policy iteration algorithm needs an exponential number of improving switches with Dantzig's rule, with Bland's rule, and with the Largest Increase pivot rule. This immediately translates to a family of linear programs for which the simplex algorithm needs an exponential number of pivot steps with the same three pivot rules. Our results yield a unified construction that simultaneously reproduces well-known lower bounds for these classical pivot rules, and we are able to infer that any (deterministic or randomized) combination of them cannot avoid an exponential worst-case behavior. Regarding the policy iteration algorithm, pivot rules typically switch multiple edges simultaneously and our lower bound for Dantzig's rule and the Largest Increase rule, which perform only single switches, seem novel. Regarding the simplex algorithm, the individual lower bounds were previously obtained separately via deformed hypercube constructions. In contrast to previous bounds for the simplex algorithm via Markov decision processes, our rigorous analysis is reasonably concise.

2012 ACM Subject Classification Theory of computation → Linear programming; Mathematics of computing → Markov processes

Keywords and phrases Bland's pivot rule, Dantzig's pivot rule, Largest Increase pivot rule, Markov decision process, policy iteration, simplex algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.27

Related Version *Full Version*: `http://arxiv.org/abs/2309.14034`

1 Introduction

Since the simplex algorithm for linear programming was proposed by Dantzig in 1951 [12], it has been a central question in discrete optimization whether it admits a polynomial time pivot rule. A positive answer to this question would yield an efficient combinatorial algorithm for solving linear programs, and thus resolve an open problem on Smale's list of mathematical problems for the 21st century [41]. It would also resolve the polynomial Hirsch conjecture [11], which states that every two vertices of every polyhedron with n facets are connected via a path of $\mathcal{O}(\text{poly}(n))$ edges. At this point, the best known pivot rules are randomized and achieve subexponential running times in expectation [21, 27, 31, 36].

For the most natural, memoryless and deterministic, pivot rules, exponential worst-case examples based on distorted hypercubes were constructed early on [4, 25, 30, 35, 38]. Amenta and Ziegler [3] introduced the notion of deformed products to unify several of these constructions. However, while this unification defines a class of polytopes that generalizes distorted hypercubes, it does not yield a unified exponential worst-case construction to exclude all pivot rules based on these deformed products, and neither does it yield new lower bounds for additional pivot rules.

Randomized and history-based pivot rules resisted similar approaches, and it was a major breakthrough in 2011 when Friedmann et al. were able to prove the first subexponential lower bound for several randomized pivot rules [20, 21, 26]. They introduced a new technique

based on a connection [39] between Howard's policy iteration algorithm [28] for Markov decision processes (MDPs) and the simplex algorithm for linear programs (LPs). The same technique was later used to prove exponential lower bounds for history-based pivot rules that had been candidates for polynomial time rules for a long time [5, 15]. While the approach via MDPs has proven powerful, the resulting analyses are often very technical (the full version of [15] with all details of the proof has 197 pages).

In this paper, we apply the MDP-based technique to classical (memoryless and deterministic) pivot rules and obtain a unified construction that excludes several pivot rules at the same time, and any combination of them, while being relatively simple.

Our results. We give a unified worst-case construction for the policy iteration algorithm for MDPs that simultaneously applies to three of the most classical pivot rules. The rigorous analysis of the resulting MDPs is reasonably concise. We note that the exponential lower bounds for Dantzig's rule and the Largest Increase rule seem novel for the considered version of the policy iteration algorithm, while the result for Bland's rule is known [37].

▶ **Theorem 1.** *There is a family $(\mathcal{D}_n)_{n\in\mathbb{N}}$ of Markov decision processes $\mathcal{D}_n$ of size $\mathcal{O}(n)$ such that policy iteration performs $\Omega(2^n)$ improving switches with Dantzig's rule, Bland's rule, and the Largest Increase pivot rule.*

In fact, all three pivot rules apply the same set of improving switches with only slight differences in the order in which they get applied. Because of this, the result still holds if we allow to change pivot rules during the course of the algorithm.

▶ **Corollary 2.** *For any (deterministic or randomized) combination of Dantzig's, Bland's, or the Largest Increase rule, the policy iteration algorithm has an exponential running time.*

A well-known connection between policy iteration and the simplex method, allows to immediately translate our result to the simplex algorithm with the same pivot rules. In particular, we obtain an exponential lower bound construction that holds even if, in every step, the entering variable is selected independently according to Dantzig's rule, Bland's rule, or the Largest Increase pivot rule, i.e., even if we change pivot rules during the course of the algorithm. In other words, we obtain a lower bound for a family of pivot rules that results from combining these three rules.

▶ **Corollary 3.** *There is a family $(\mathcal{L}_n)_{n\in\mathbb{N}}$ of linear programs $\mathcal{L}_n$ of size $\mathcal{O}(n)$ such that the simplex algorithm performs $\Omega(2^n)$ pivot operations for any (deterministic or randomized) combination of Dantzig's, Bland's, or the Largest Increase pivot rule.*

Related work. Policy iteration for MDPs has been studied extensively for a variety of pivot rules. In its original version [28], the algorithm applies improving switches to the current policy in all states simultaneously in every step. Fearnley [18] showed an exponential lower bound for a greedy pivot rule that selects the best improvement in every switchable state. In this paper, we focus on pivot rules that only apply a single switch in each iteration. Most of the MDP constructions for randomized or history-based pivot rules [5, 15, 21, 26] consider this case, and Melekopoglou and Condon [37] gave exponential lower bounds for several such deterministic pivot rules. We emphasize that their constructions already include an exponential lower bound for Bland's rule [8]. Since policy iteration is traditionally considered with simultaneous switches, to the best of our knowledge, no exponential lower bounds are known for Dantzig's rule [12] and the Largest Increase rule [11] in the setting of single switches.

There is a strong connection between policy iteration and the simplex algorithm, which, under certain conditions (see below), yields that worst-case results for policy iteration carry over to the simplex method [39]. This connection was used to establish subexponential lower bounds for randomized pivot rules, namely Randomized Bland [26] and Random-Edge, RaisingTheBar and Random-Facet [21]. It also lead to exponential lower bounds for history-based rules, namely Cunningham's rule [5] and Zadeh's rule [15]. Conversely, lower bounds for the simplex algorithm with classical pivot rules were obtained via deformed hypercubes [3] and do not transfer to MDPs. Such results include lower bounds for Dantzig's rule [35], the Largest Increase rule [30], Bland's rule [4], the Steepest Edge rule [25], and the Shadow Vertex rule [38]. We provide an alternate lower bound construction for the first three of these rules via a family of MDPs. As far as we can tell, as a side product, this yields the first exponential lower bound for policy iteration with Dantzig's rule and the Largest Increase rule.

While it remains open whether LPs can be solved in strongly polynomial time, there are several, both deterministic [32, 34] and randomized [7, 17, 33], algorithms that solve LPs in weakly polynomial time. A (strongly) polynomial time pivot rule for the simplex algorithm would immediately yield a strongly polynomial algorithm.

There have been different attempts to deal with the worst-case behavior of the simplex method from a theoretical perspective. For example, the excessive running time was justified by showing that the simplex algorithm with Dantzig's original pivot rule is *NP-mighty* [16], which means that it can be used to solve NP-hard problems. This result was subsequently strengthened by establishing that deciding which solution is computed and whether a given basis will occur is PSPACE-complete [2, 19]. On the positive side, there are different results explaining the efficiency of the simplex method in practice, such as average-case analyses [1, 9, 44]. Spielman and Teng [42] introduced smoothed analysis as a way of bridging the gap between average-case and worst-case analysis. They showed that the simplex algorithm with the shadow vertex pivot rule [24] has a polynomial smoothed complexity, and their results were further improved later [10, 14, 29, 45].

Another approach to derive stronger lower bounds on pivot rules is to consider combinatorical abstractions of LPs, such as Unique Sink Orientations (USOs) [23]. There is still a large gap between the best known deterministic algorithm for finding the unique sink, which is exponential [43], and the almost quadratic lower bound [40]. Considering randomized rules, the Random-Facet pivot rule, which is the best known simplex rule [27], is also the best known pivot rule for acyclic USOs [22], achieving a subexponential running time in both settings.

2 Preliminaries

Markov Decision Processes

A *Markov decision process* is an infinite duration one-player game on a finite directed graph $G = (V_A, V_R, E_A, E_R, r, p)$. The vertex set $V = V_A \cup V_R$ of the graph is divided into *agent vertices* V_A and *randomization vertices* V_R. Every agent edge $e \in E_A \subseteq V_A \times V$ is assigned a *reward* $r(e) \in \mathbb{R}$, while every randomization edge $\hat{e} \in E_R \subseteq V_R \times V$ is assigned a *transition probability* $p(\hat{e}) \in [0, 1]$. Outgoing transition probabilities add to one in every randomization vertex.

A process starts in an arbitrary starting vertex. If this is an agent vertex, the agent moves along one of the outgoing edges of this vertex (we assume that all vertices have at least one outgoing edge) and collects the corresponding reward. Otherwise, it gets randomly moved along one of the outgoing edges according to the transition probabilities. The process continues in this manner ad infinitum.

An agent vertex $s \in V_A$ whose only outgoing edge is a self-loop with reward zero is called *sink* of G if it is reachable from all vertices. A *policy* for G is a function $\pi\colon V_A \to V$ with $(v, \pi(v)) \in E_A$ for all $v \in V_A$, determining the behavior of the process in agent vertices. A policy π for G is called *weak unichain* if G has a sink s such that π reaches s with a probability of one from every starting vertex.

The *value* of a vertex v w.r.t. a policy π for a Markov decision process G is given by the expected total reward that the agent collects with policy π when the process starts in v. More formally, the value function $\mathrm{Val}_{\pi,G}\colon V \to \mathbb{R}$ is defined by the following system of Bellman [6] equations

$$\mathrm{Val}_{\pi,G}(u) = \begin{cases} r((u,\pi(u))) + \mathrm{Val}_{\pi,G}(\pi(u)), & \text{if } u \in V_A, \\ \sum\limits_{v \in \Gamma^+(u)} p((u,v))\,\mathrm{Val}_{\pi,G}(v), & \text{if } u \in V_R, \end{cases}$$

together with $\mathrm{Val}_{\pi,G}(s) = 0$ if G has a sink s. The policy π is optimal (w.r.t. the *expected total reward criterion*) if $\mathrm{Val}_{\pi,G}(v) \geq \mathrm{Val}_{\tilde{\pi},G}(v)$ for all $v \in V_A$ and all policies $\tilde{\pi}$ for G. Whenever the underlying process G is clear from the context, we write Val_π instead of $\mathrm{Val}_{\pi,G}$.

We say that the agent edge $(u,v) \in E_A$ is an *improving switch* for the policy π for process G if it satisfies $z_{\pi,G}(u,v) \coloneqq r((u,v)) + \mathrm{Val}_{\pi,G}(v) - \mathrm{Val}_{\pi,G}(u) > 0$, where $z_{\pi,G}(u,v)$ are the *reduced costs* of (u,v) with respect to π. Again, we usually write z_π instead of $z_{\pi,G}$.

If we *apply* an improving switch $s = (u,v) \in E_A$ to a policy π, we obtain a new policy π^s which is given by $\pi^s(u) = v$ and $\pi^s(w) = \pi(w)$ for all $w \in V_A \setminus \{u\}$. The improving switch s increases the value of u without decreasing the value of any other vertex.

Policy Iteration for Markov Decision Processes

Howard's [28] policy iteration algorithm receives as input a finite Markov decision process G and a weak unichain policy π for G. It then iteratively applies a set of improving switches to the current policy until there are none left. In the remainder of this paper, we consider a version of this algorithm that applies a single switch in every iteration, cf. Algorithm 1. Due to monotonicity of the vertex values, this procedure visits every policy at most once. As there are only finitely many policies, the algorithm thus terminates after a finite number of iterations for every initial policy.

Algorithm 1 PolicyIteration(G, π).

input: a weak unichain policy π for a Markov decision process G

while π *admits an improving switch*:
  $\bar{s} \leftarrow$ improving switch for π
  $\pi \leftarrow \pi^{\bar{s}}$
return π

We know that the policy iteration algorithm returns an optimal policy if there is an optimal policy which is weak unichain.

▶ **Theorem 4** ([20]). *Let π be a weak unichain policy for a Markov decision process G. If G admits a weak unichain, optimal policy, then* PolicyIteration(G, π) *only visits weak unichain policies and returns an optimal policy w.r.t. the expected total reward criterion.*

In this paper, we consider the following three *pivot rules*, i.e., rules that determine the choice of PolicyIteration(G, π) in each iteration:

- *Bland's pivot rule* assigns a unique number to every agent edge of G. Then, in every iteration, it chooses the improving switch with the smallest number.
- *Dantzig's pivot rule* chooses an improving switch $\bar{s}$ maximizing the reduced costs $z_\pi(\bar{s})$.
- The *Largest Increase rule* chooses an improving switch $\bar{s}$ maximizing $\sum_{v \in V_A} \mathrm{Val}_{\pi^{\bar{s}}}(v)$.

A Connection between Policy Iteration and the Simplex Method

Given a Markov decision process, we can formulate a linear program such that the application of the simplex method is in some sense equivalent to the application of policy iteration. We refer to [20] for more details and the derivation of the following result.

▶ **Theorem 5** ([20]). *Let π be a weak unichain policy for a Markov decision process G. Assume that there is an optimal, weak unichain policy for G and that* PolicyIteration(G, π) *with a given pivot rule takes N iterations. Then, there is an LP of linear size such that the simplex algorithm with the same pivot rule takes N iterations.*

In terms of the simplex method, Bland's pivot rule chooses the entering variable of smallest index [8], Dantzig's rule chooses an entering variable maximizing the reduced costs [12], and the Largest Increase rule greedily maximizes the objective function value.

The linear program in the previous theorem has one variable for every agent edge of the Markov decision process such that the reduced costs of a given edge equal the reduced costs of the corresponding variable, and the objective function equals the sum over all vertex values as given in the Largest Increase rule for policy iteration [5, 15, 26]. Therefore, the choices of each pivot rule in the two settings are consistent.

Additionally, we want to mention that the linear program from Theorem 5 is always non-degenerate. Therefore, we cannot reduce the number of required iterations on these programs by combining a given pivot rule with the Lexicographic pivot rule [13].

Notation

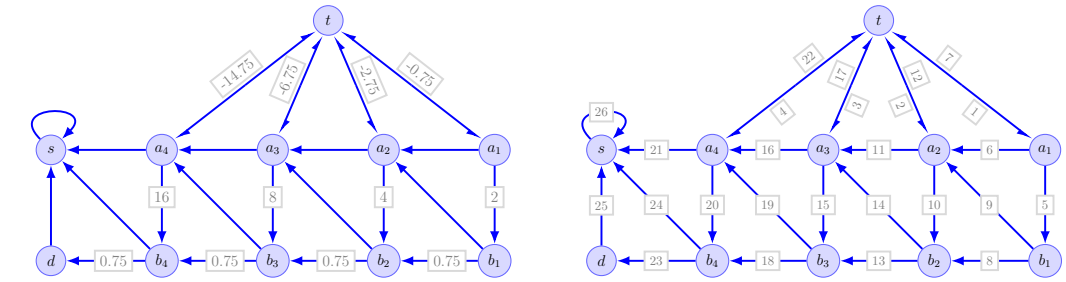
Let $n \in \mathbb{N}$ be fixed. We write $[n] = \{1, 2, \dots, n\}$ and $[n]_0 = \{0, 1, \dots, n\}$. Then, the set of all numbers that can be represented with n bits is $[2^n - 1]_0$.

For every $x \in [2^n - 1]_0$ and $i \in [n]$, let x_i denote the i-th bit of x, i.e., $x = \sum_{i \in [n]} x_i 2^{i-1}$, and let $L(i, x) = \max\{j \in [i-2] \mid x_j = 1 \text{ or } j = 1\}$ for $i \geq 3$. Finally, for $x \in [2^n - 1]$, we denote the *least significant set bit* of x by $\ell_1(x) = \min\{i \in [n] : x_i = 1\}$, and the *most significant set bit* of x by $\mathrm{m}_1(x) = \max\{i \in [n] : x_i = 1\}$.

Let $G = (V_A, V_R, E_A, E_R, r, p)$ be a Markov decision process. For $v \in V_A \cup V_R$, we write $\Gamma^+_G(v) = \{w \in V_A \cup V_R \colon (v, w) \in E_A \cup E_R\}$. If the underlying process is clear from the context, we just write $\Gamma^+(v)$.

3 An Exponential Lower Bound for Bland's pivot rule

In this section, we consider a family $(\mathcal{B}_n = (V_{\mathcal{B}_n}, E_{\mathcal{B}_n}, r_{\mathcal{B}_n}))_{n \in \mathbb{N}}$ of Markov decision processes, which do not involve any randomization. Consider Figure 1a for a drawing of $\mathcal{B}_4$. Every process $\mathcal{B}_n$ consists of n separate levels, together with a global *transportation vertex* t, a sink s, and a *dummy vertex* d. Each level $\ell \in [n]$ comprises two vertices, called a_ℓ and b_ℓ. For convenience, we sometimes denote the sink by a_{n+1} and the dummy vertex by b_{n+1}.



(a) (b)

Figure 1 Two drawings of the Markov decision process $\mathcal{B}_4$. In (a), edge labels denote rewards and unlabeled edges have a reward of zero. In (b), edge labels define the Bland numbering $\mathcal{N}_{\mathcal{B}_4}$.

Table 1 Edge names and the definition of the Bland numbering $\mathcal{N}_{\mathcal{B}_n}$, where $i \in [n]$.

$e \in E_{\mathcal{B}_n}$		$\mathcal{N}_{\mathcal{B}_n}(e)$
(t,a_i)	travel(i)	i
(a_i, b_i)	enter(i)	$n+1+5(i-1)$
(a_i, a_{i+1})	skip(i)	$n+2+5(i-1)$
(a_i, t)	board(i)	$n+3+5(i-1)$
(b_i, b_{i+1})	stay(i)	$n+4+5(i-1)$
(b_i, a_{i+1})	leave(i)	$n+5+5(i-1)$

In vertex a_ℓ, the agent can either *enter* level ℓ by going to vertex b_ℓ, *skip* this level by going to vertex $a_{\ell+1}$, or *board* the transportation vertex by going to t. From the transportation vertex, the agent *travels* to one of the vertices a_i with $i \in [n]$. In b_ℓ, the agent can decide between *leaving* the set $\bigcup_{i\in[n+1]}\{b_i\}$ by going to $a_{\ell+1}$ and *staying* in this set by going to $b_{\ell+1}$. We will simply say that the agent leaves level ℓ or stays in level ℓ, respectively.

Finally, when the agent reaches the dummy vertex d, it must go to the sink, and the only outgoing edge of the sink s is the self-loop (s,s).

The function $r_{\mathcal{B}_n}$ grants the agent a reward of 2^ℓ for entering level ℓ, a reward of 0.75 for staying in level ℓ, and a (negative) reward of $(-2^\ell + 1.25)$ for boarding t from a_ℓ; all other rewards are zero.

The *Bland numbering* $\mathcal{N}_{\mathcal{B}_n} : E_{\mathcal{B}_n} \to |E_{\mathcal{B}_n}|$ of the edges of $\mathcal{B}_n$ is defined in Table 1, together with $\mathcal{N}_{\mathcal{B}_n}((d,s)) = 6n+1$ and $\mathcal{N}_{\mathcal{B}_n}((s,s)) = 6n+2 (= |E_{\mathcal{B}_n}|)$. This table also contains alternative names for the edges, which match the description above and which we will use to simplify the exposition. Consider Figure 1b for the Bland numbering of $\mathcal{B}_4$.

In the following, consider $\mathcal{B}_n$ for some arbitrary but fixed $n \in \mathbb{N}$. The aim of this section is to show that PolicyIteration with Bland's pivot rule, cf. Algorithm 2, applies $\Omega(2^n)$ improving switches when given $\mathcal{B}_n$, a suitable initial policy, and $\mathcal{N}_{\mathcal{B}_n}$ as input.

Algorithm 2 Bland$(G, \pi, \mathcal{N})$.

input: Markov decision process G, weak unichain policy π, edge numbering $\mathcal{N}$

while π *admits an improving switch*:
  $\bar{s} \leftarrow$ the improving switch s for π that minimizes $\mathcal{N}(s)$
  $\pi \leftarrow \pi^{\bar{s}}$
return π

More precisely, we will see that the algorithm visits all of the following policies.

▶ **Definition 6.** The policy π_0 for $\mathcal{B}_n$ such that travel(1) is active, and skip(i) and leave(i) are active for all $i \in [n]$ is the *canonical policy* for 0. For $x \in [2^n - 1]$, the policy π_x for $\mathcal{B}_n$ is the *canonical policy* for x if it satisfies the following conditions:

<1> The policy travels from t to the least significant set bit, i.e., travel($\ell_1(x)$) is active.

<2> It collects no reward above the most significant set bit, i.e., leave($\mathrm{m}_1(x)$), skip(i), and leave(i) are active for all $\mathrm{m}_1(x) < i \leq n$.

<3> Every set bit $x_i = 1$ determines the behavior of the policy down to the next, less significant set bit or, if $i = \ell_1(x)$, down to the first bit:

- <a> enter(i) is active.
- <b> if $i = 2$, then leave(1) is active. If additionally $x_1 = 0$, then skip(1) is active.
- <c> if $i \geq 3$ and $x_{i-1} = 1$, then leave($i-1$) is active.
- <d> if $i \geq 3$ and $x_{i-1} = 0$:
  - <$\mathbf{d_1}$> stay($i-1$), skip($i-1$), and leave($i-2$) are active.
  - <$\mathbf{d_2}$> if $L(i,x) < i-2$, then for all $j \in \{L(i,x)+1, \dots, i-2\}$, the edges board($j$) and stay($j-1$) are active; if $L(i,x) = 1$ and $x_1 = 0$, then board(1) is active. ⌟

Consider Figure 2a and Figure 2d for examples of canonical policies. Note that canonical policies exist and are unique as the definition contains precisely one condition on every agent vertex with more than one outgoing edge. Further, the 2^n canonical policies are pairwise different as enter(i) is active in π_x if and only if $x_i = 1$.

We will now analyze the behavior of $\textsc{Bland}(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$, i.e., we choose the canonical policy for zero as our initial policy. Since this policy visits every vertex except the sink only once, it is weak unichain.

▶ **Observation 7.** *The canonical policy π_0 is a weak unichain policy for $\mathcal{B}_n$.*

Thus, according to Theorem 5, the following result will allow us to transfer our results for the policy iteration algorithm to the simplex method.

▶ **Lemma 8.** *Let the policy π_* for $\mathcal{B}_n$ be determined as follows:* stay(n) *and* travel(1) *are active,* enter(i) *is active for all $i \in [n]$, and* leave(j) *is active for all $j \in [n-1]$. Then, π_* is weak unichain and optimal for $\mathcal{B}_n$.*

Proof. Since π_* visits every vertex, besides the sink, only once, it is weak unichain. For optimality, note that t travels to a_1 and that, when starting in a vertex a_ℓ, policy π_* enters level ℓ and all levels above and collects the reward of stay(n). The policy is thus clearly optimal among the set of policies that do not use boarding edges.

Further, we have $r_{\mathcal{B}_n}(\text{board}(\ell)) = -2^\ell + 1.25 = -(\sum_{i=1}^{\ell-1} 2^i + 0.75)$. That is, the costs of board(ℓ) equal the maximum reward that can be collected in the first $\ell - 1$ levels. Thus, we cannot increase vertex values by using boarding edges, which yields that π_* is optimal. ◀

The following technical result will be helpful in the upcoming proofs.

▶ **Lemma 9.** *Let $x \in [2^n - 1]_0$ and $i \in [n]$. Then,* travel(i) *is not improving for π_x.*

Proof. All vertex values with respect to π_0 are zero, and $r_{\mathcal{B}_n}(\text{travel}(i)) = 0$. Thus, the claim holds for $x = 0$, so we assume $x \in [2^n - 1]$ in the following.

Let the vertices a_k and a_ℓ either correspond to successive set bits, i.e., $x_k = x_\ell = 1$ and $x_j = 0$ for all $k < j < \ell$, or let $k = \mathrm{m}_1(x)$ and $\ell = n + 1$. Either way, Definition 6 implies that π_x includes a path from a_k to a_ℓ, which does not contain any boarding edge.

Hence, we have $\mathrm{Val}_{\pi_x}(a_\alpha) \geq \mathrm{Val}_{\pi_x}(a_\beta) \geq 0$ for all set bits $x_\alpha = x_\beta = 1$ with $\alpha \leq \beta$. Since the transportation vertex chooses the least significant set bit in π_x, this yields that travel(i) is not improving if $x_i = 1$.

Further, Definition 6 yields that $x_j = 1$ if and only if enter(j) is active in π_x. Thus, when starting in some vertex a_i with $x_i = 0$, policy π_x either boards t from a_i or it skips levels until reaching a node that boards t, a level corresponding to a set bit, or the sink. In all four cases, travel(i) is not improving. This completes the proof. ◀

We will show in two steps that, when using the initial policy π_0, Bland visits all of the other canonical policies. Firstly, given the canonical policy for an arbitrary even integer x, we see that the algorithm applies improving switches until reaching the canonical policy π_{x+1}.

▶ **Lemma 10.** *Let $x \in [2^n - 2]_0$ be even. Then, Bland$(\mathcal{B}_n, \pi_x, \mathcal{N}_{\mathcal{B}_n})$ visits π_{x+1}.*

Proof. According to Lemma 9, no travel edges are improving for π_x, so the Bland numbering $\mathcal{N}_{\mathcal{B}_n}$ yields that the algorithm applies the switch enter(1) to π_x if it is improving. This edge is improving for π_0, and one can easily check that its application results in the canonical policy π_1. Hence, it suffices to consider $x \neq 0$ in the following. This yields $\ell_1 \coloneqq \ell_1(x) > 1$ as x is even. Influenced by condition <3> from Definition 6, we consider two cases.

Firstly, if $\ell_1 = 2$, conditions <1> and <b> state that travel(2), skip(1) and leave(1) are active in π_x. Hence, enter(1) is improving for π_x and gets applied by Bland. The edge travel(1) becomes improving and gets applied next as it minimizes $\mathcal{N}_{\mathcal{B}_n}$.

Secondly, if $\ell_1 \geq 3$, conditions <1> and <d> yield that π_x includes the paths (a_1, t, a_{ℓ_1}) and $(b_1, b_2, \ldots, b_{\ell_1-2}, a_{\ell_1-1}, a_{\ell_1}) =: P$. Hence, $\mathrm{Val}_{\pi_x}(a_1) \leq \mathrm{Val}_{\pi_x}(a_{\ell_1}) \leq \mathrm{Val}_{\pi_x}(b_1)$. Therefore, as enter(1) has a positive reward, it is improving and gets applied to π_x. The new policy walks from a_1 to b_1 and then follows the path P, so a_1 has a higher value than a_{ℓ_1}. Since travel(ℓ_1) is active in π_x, the edge travel(1) is improving and gets applied next.

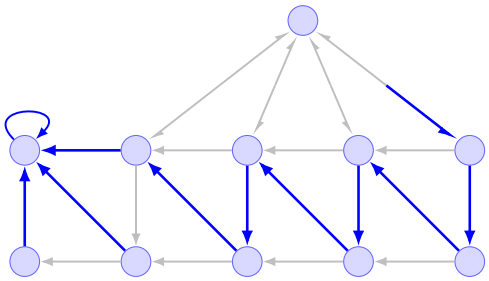
Let π denote the policy resulting from the application of enter(1) and travel(1) to π_x. It now suffices to show that π satisfies the conditions of Definition 6 for $x + 1$.

As $x + 1$ is odd, we have $\ell_1(x + 1) = 1$, so π satisfies the first condition. Further, the second condition remains satisfied as both applied switches are below the most significant set bit. Finally, the third condition now requires that enter(1) is active – instead of skip(1) if $\ell_1 = 2$, or board(1) if $\ell_1 \geq 3$ – and otherwise contains the same requirements. Hence, π is the canonical policy for $x + 1$. ◀

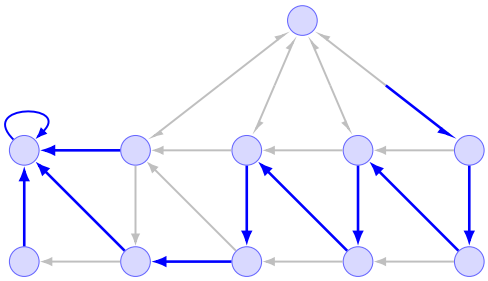
Secondly, we need to show that the algorithm also transforms the canoncial policy π_x for an arbitrary odd number x into the next canonical policy π_{x+1}. We will see that the algorithm does this by applying the following sequence of improving switches.

▶ **Definition 11.** Let $x \in [2^n - 3]$ be odd and write $\ell \coloneqq \ell_0(x) > 1$. Then, the *canonical phases* with respect to x are:

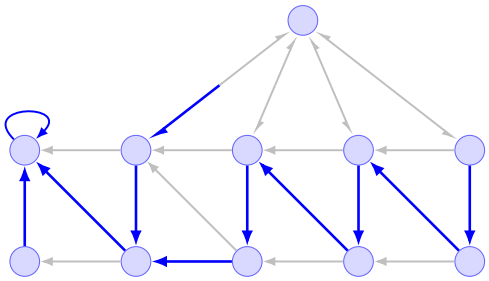
1. If $x_{\ell+1} = 1$, activate leave(ℓ).
2. If $x_{\ell+1} = 1$ or $\ell > \mathrm{m}_1(x)$, activate stay($\ell - 1$).
3. Activate enter(ℓ) and travel(ℓ).
4. If $\ell \geq 3$, activate board(j) for all $j \in [\ell - 2]$ in increasing order.
5. Activate skip($\ell - 1$).
6. If $\ell \geq 4$, activate stay(j) for all $j \in [\ell - 3]$ in decreasing order.
7. If $\ell = 2$, activate leave(1). ⌟



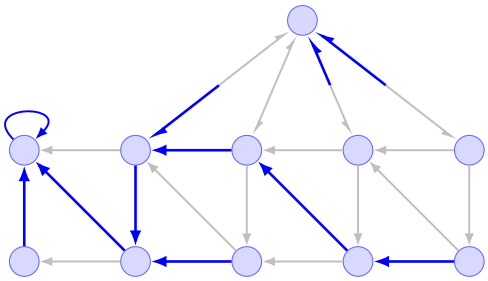
(a) The canonical policy π_7 for $\mathcal{B}_4$.



(b) The policy that results from applying the first two canonical phases w.r.t. 7 to π_7.



(c) The policy that results from applying the first three canonical phases w.r.t. 7 to π_7.



(d) The canonical policy π_8 for $\mathcal{B}_4$, which results from applying all canonical phases w.r.t. 7 to π_7.

Figure 2 An example that illustrates how the canonical phases transform one canonical policy into the next one. Active edges are depicted in a bold blue color, while inactive edges are slightly transparent. Note that π_7 enters the first three levels, which correspond to the set bits in the binary representation of 7; analogously, π_8 only enters the fourth level.

The following Lemma shows that if the algorithm applies the canonical phases to the corresponding canonical policy, it reaches the next canonical policy.[1] Consider Figure 2 for an example.

▶ **Lemma 12.** *Let π_x be the canonical policy for some odd $x \in [2^n - 3]$. Applying the canonical phases with respect to x to π_x results in the canonical policy π_{x+1}.*

Finally, we show that Bland actually applies the canonical phases when given the corresponding canonical policy.

▶ **Lemma 13.** *Let $x \in [2^n - 3]$ be odd. Then, $\textsc{Bland}(\mathcal{B}_n, \pi_x, \mathcal{N}_{\mathcal{B}_n})$ visits π_{x+1}.*

According to Lemma 10, Bland transforms every even canonical policy π_x into π_{x+1}, and by Lemma 13, the same holds for odd canonical policies. Since the initial policy is canonical for zero, this yields that $\textsc{Bland}(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$ visits all canonical policies π_i with $i \in [2^n - 1]$. Since these are pairwise different, this proves the main result of this section.

▶ **Theorem 14.** *There is an initial policy such that the policy iteration algorithm with Bland's pivot rule performs $\Omega(2^n)$ improving switches when applied to $\mathcal{B}_n$.*

We close this section with two technical observations that help us later.

▶ **Observation 15.** *For every $i \in [n]$, whenever $\textsc{Bland}(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$ applies the improving switch $\text{skip}(i)$, this edge has higher reduced costs than $\text{board}(i)$; whenever it applies $\text{enter}(i)$, this edge has higher reduced costs than $\text{skip}(i)$ and $\text{board}(i)$.*

▶ **Observation 16.** *At any point during the execution of $\textsc{Bland}(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$, at most one of the edges $\text{travel}(i)$ with $i \in [n]$ is improving.*

[1] All missing proofs can be found in the full version.

4 A Combined Exponential Bound

In this section, we consider a family $(\mathcal{D}_n = (V_n^A, V_n^R, E_n^A, E_n^R, r_{\mathcal{D}_n}, p_{\mathcal{D}_n}))_{n\in\mathbb{N}}$ of Markov decision processes such that each process $\mathcal{D}_n$ results from the process $\mathcal{B}_n = (V_{\mathcal{B}_n}, E_{\mathcal{B}_n}, r_{\mathcal{B}_n})$ of the previous section by replacing every edge, besides the sink-loop, with the construction given in Figure 3; note that the construction uses a probability $p_v \in (0, 1]$ for every $v \in V_{\mathcal{B}_n} \setminus \{s\}$, which we will choose later.

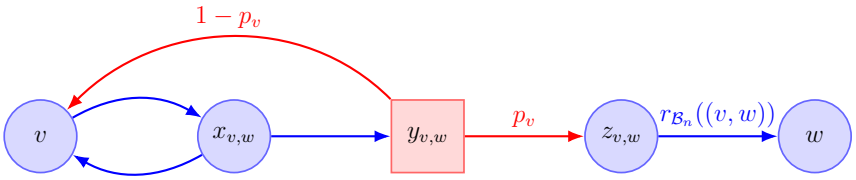


Figure 3 The construction that replaces every edge $(v, w) \in E_{\mathcal{B}_n} \setminus \{(s, s)\}$ in $\mathcal{D}_n$. Circular vertices are agent vertices, square ones are randomization vertices. Edge labels denote rewards and probabilities, where $p_v \in (0, 1]$. Note that, since v and w remain in the process, we have $V_{\mathcal{B}_n} \subseteq V_n^A$.

In the following, consider $\mathcal{D}_n$ for some arbitrary but fixed $n \in \mathbb{N}$. The aim of this section is to show that policy iteration with Bland's rule, with Dantzig's rule, and with the Largest Increase rule performs $\Omega(2^n)$ improving switches to a suitable initial policy for $\mathcal{D}_n$.

Before we can analyze the behavior of Bland on $\mathcal{D}_n$, we need to specify the Bland numbering $\mathcal{N}_{\mathcal{D}_n} : E_n^A \to |E_n^A|$ for $\mathcal{D}_n$. It is constructed as follows: starting from the numbering $\mathcal{N}_{\mathcal{B}_n}$, replace every edge $(v, w) \in E_{\mathcal{B}_n} \setminus \{(s, s)\}$ by the edges $(x_{v,w}, y_{v,w})$ and $(v, x_{v,w})$. Then, insert all edges of the form $(x_{u,\cdot}, u)$ with $u \in V_{\mathcal{B}_n} \setminus \{s\}$ at the beginning of the numbering (the internal order of these edges can be chosen arbitrarily). We do not need to specify the Bland numbers of edges that are the unique outgoing edge of a vertex.

Now that we have a Bland numbering, we want to transfer our results from the previous section to the new Markov decision process $\mathcal{D}_n$. The following definition extends policies for $\mathcal{B}_n$ to policies for $\mathcal{D}_n$.

▶ **Definition 17.** Let π and π' be policies for $\mathcal{B}_n$ and $\mathcal{D}_n$, respectively, and let $v \in V_{\mathcal{B}_n} \setminus \{s\}$. Assume there is a $w \in \Gamma^+_{\mathcal{B}_n}(v)$ such that $(v, x_{v,w})$, $(x_{v,w}, y_{v,w})$, and $(x_{v,u}, v)$ are active in π' for all $u \in \Gamma^+_{\mathcal{B}_n}(v) \setminus \{w\}$. Then, we say that v is *(w-)oriented w.r.t. π'*. We call π' the *twin policy* of π if every vertex $v \in V_{\mathcal{B}_n} \setminus \{s\}$ is $\pi(v)$-oriented w.r.t. π'. ◿

Let π'_0 denote the twin policy of the canonical policy π_0 for $\mathcal{B}_n$. We could start by showing that $\textsc{Bland}(\mathcal{D}_n, \pi'_0, \mathcal{N}_{\mathcal{D}_n})$ visits the twin policy of every policy that $\textsc{Bland}(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$ visits. Note that this would immediately imply the desired exponential number of improving switches. However, we prefer to gather some general results first, which then allows for a more unified treatment of the three pivot rules.

Starting in a w-oriented vertex $v \in V_{\mathcal{B}_n} \setminus \{s\}$, the agent reaches vertex w with probability one (due to $p_v > 0$), while collecting a reward of $r_{\mathcal{B}_n}((v, w))$. This immediately yields the following result.

▶ **Observation 18.** *Let π be a policy for $\mathcal{B}_n$ with twin policy π' for $\mathcal{D}_n$. Then, for every vertex $v \in V_{\mathcal{B}_n}$, we have* $\mathrm{Val}_{\pi,\mathcal{B}_n}(v) = \mathrm{Val}_{\pi',\mathcal{D}_n}(v)$.

By the same argument, twin policies of weak unichain policies are weak unichain, and the proof idea of Lemma 8 carries over.

▶ **Observation 19.** *The twin policy of every weak unichain policy for $\mathcal{B}_n$ is a weak unichain policy for $\mathcal{D}_n$. The twin policy of the optimal policy for $\mathcal{B}_n$ is optimal for $\mathcal{D}_n$.*

By Theorem 4, this guarantees the correctness of PolicyIteration$(\mathcal{D}_n, \pi'_0)$. Further, Theorem 5 will allow us to carry our results over to the simplex method.

Since twin policies are central in our analysis, it comes in handy that only a certain type of edges might be improving for them.

▶ **Observation 20.** *Let π' be the twin policy of some policy for $\mathcal{B}_n$. Then, all improving switches for π' are of the form $(x_{v,w}, y_{v,w}) \in E_n^A$ for some $(v,w) \in E_{\mathcal{B}_n}$.*

Proof. Since every vertex $u \in V_{\mathcal{B}_n} \setminus \{s\}$ is oriented w.r.t π', edges of the form $(x_{u,\cdot}, u)$ or $(u, x_{u,\cdot})$ are either active or their application creates a zero-reward cycle of length two. Hence, none of these edges is improving for π'. ◀

The following Lemma shows how the probabilities $(p_v)_{v \in V_{\mathcal{B}_n} \setminus \{s\}}$ affect the reduced costs of these potentially improving edges. Further, it yields a connection between the improving switches for a policy for $\mathcal{B}_n$ and those for its twin policy.

▶ **Lemma 21.** *Let π be a policy for $\mathcal{B}_n$ with twin policy π', and let $(v,w) \in E_{\mathcal{B}_n} \setminus \{(s,s)\}$. Then, $z_{\pi',\mathcal{D}_n}(x_{v,w}, y_{v,w}) = p_v \cdot z_{\pi,\mathcal{B}_n}(v,w)$. In particular, $(x_{v,w}, y_{v,w})$ is improving for π' if and only if (v,w) is improving for π.*

Proof. For convenience, we write x, y, and z instead of $x_{v,w}$, $y_{v,w}$, and $z_{v,w}$. If (v,w) is active in π, vertex v is w-oriented w.r.t. π'. Thus, (x,y) is active in π' as well. Hence, both edges are not improving as they have reduced costs of $z_{\pi'}(x,y) = z_\pi(v,w) = 0$.

Now assume that (v,w) is inactive in π, which yields that (x,v) is active in π'. We obtain

$$\begin{aligned} z_{\pi',\mathcal{D}_n}(x,y) &= \mathrm{Val}_{\pi'}(y) - \mathrm{Val}_{\pi'}(x) = \mathrm{Val}_{\pi'}(y) - \mathrm{Val}_{\pi'}(v) \\ &= p_v \,\mathrm{Val}_{\pi'}(z) + (1-p_v)\,\mathrm{Val}_{\pi'}(v) - \mathrm{Val}_{\pi'}(v) \\ &= p_v(\mathrm{Val}_{\pi'}(w) + r_{\mathcal{B}_n}((v,w)) - \mathrm{Val}_{\pi'}(v)) = p_v z_{\pi,\mathcal{B}_n}(v,w), \end{aligned} \tag{1}$$

where we used Observation 18 for the last equality. The equivalence holds since $p_v > 0$. ◀

Note that we can transform a given twin policy with three switches into a different one by changing the orientation of an agent vertex $v \in V_{\mathcal{B}_n} \setminus \{s\}$. The following Lemma shows that, if applied consecutively, these switches all have the same reduced costs.

▶ **Lemma 22.** *Let $(v,w) \in E_{\mathcal{B}_n} \setminus \{(s,s)\}$ and let the policy π for $\mathcal{D}_n$ be the twin policy of some weak unichain policy for $\mathcal{B}_n$. If the edge $(x_{v,w}, y_{v,w})$ is improving for π, we have*

$$z_\pi(x_{v,w}, y_{v,w}) = z_{\pi'}(v, x_{v,w}) = z_{\pi''}(\pi(v), v),$$

where π' denotes the policy that results from applying $(x_{v,w}, y_{v,w})$ to π and π'' denotes the policy that results from applying $(v, x_{v,w})$ to π'.

It is essential for the proofs of Lemma 10 and Lemma 13 that Bland$(\mathcal{B}_n, \mathcal{N}_{\mathcal{B}_n})$ prefers switches in vertices appearing early in the *vertex numbering* $\mathcal{N}_V \colon V_{\mathcal{B}_n} \to |V_{\mathcal{B}_n}|$ given by $(t, a_1, b_1, a_2, b_2, \ldots, a_n, b_n, d, s)$, i.e., let $\mathcal{N}_V(t) = 1$, $\mathcal{N}_V(a_1) = 2$, and so on. Using the following definition, we can observe a similar behavior of policy iteration with Dantzig's rule, cf. Algorithm 3, on $\mathcal{D}_n$.

▶ **Definition 23.** The edge $e \in E_n^A$ *belongs* to vertex $v \in V_{\mathcal{B}_n} \setminus \{s\}$ if

$$e \in \mathrm{B}(v) \coloneqq \bigcup_{w \in \Gamma^+_{\mathcal{B}_n}(v)} \{(x_{v,w}, y_{v,w}), (v, x_{v,w}), (x_{v,w}, v)\}.$$ ⌟

We obtain the following bounds on the reduced costs.

▶ **Lemma 24.** *Let $v \in V_{\mathcal{B}_n} \setminus \{s\}$ and $e \in B(v)$ be arbitrary. Let π be a weak unichain policy for $\mathcal{D}_n$ such that all vertex values w.r.t. π are non-negative. If e is improving for π, then its reduced costs are bounded by $p_v \cdot 0.25 \leq z_\pi(e) \leq p_v \cdot 2^{n+2}$.*

Proof. Since $e \in B(v)$, we have $e \in \{(x_{v,w}, y_{v,w}), (v, x_{v,w}), (x_{v,w}, v)\}$ for some $w \in \Gamma^+_{\mathcal{B}_n}(v)$. For convenience, we write x, y, and z instead of $x_{v,w}$, $y_{v,w}$, and $z_{v,w}$.

Firstly, assume that $e = (x, y)$. Then, since e is improving, (x, v) is active in π. As in equation (1), we obtain $z_\pi(x, y) = p_v(\mathrm{Val}_\pi(w) + r_{\mathcal{B}_n}((v, w)) - \mathrm{Val}_\pi(v)) =: p_v \cdot \delta(\pi, v, w)$.

Secondly, assume that $e = (v, x)$. Then, (x, y) is active in π as otherwise e would not be improving due to $z_\pi(v, x) = \mathrm{Val}_\pi(x) - \mathrm{Val}_\pi(v) = 0$. This yields

$$z_\pi(v, x) = \mathrm{Val}_\pi(x) - \mathrm{Val}_\pi(v) = \mathrm{Val}_\pi(y) - \mathrm{Val}_\pi(v) = p_v \cdot \delta(\pi, v, w). \tag{2}$$

Lastly, assume $e = (x, v)$. Then, as before, (x, y) is active in π. We can thus conclude from (2) that $z_\pi(x, v) = \mathrm{Val}_\pi(v) - \mathrm{Val}_\pi(x) = -p_v \cdot \delta(\pi, v, w)$.

By assumption, every vertex has a non-negative value with respect to π. Further, all vertex values are bounded from above by the maximum vertex value w.r.t. the optimal policy for $\mathcal{D}_n$. By Lemma 8, Observation 18, and Observation 19, this is $\mathrm{Val}_{\pi'_*}(t) = 2^{n+1} - 1.25$.

Since the absolute value of any edge reward is at most 2^n, we obtain

$$|\delta(\pi, v, w)| \leq \left(2^{n+1} - 1.25 + 2^n\right) \leq 2^{n+2}.$$

Hence, we have an upper bound of $z_\pi(e) \leq p_v \cdot 2^{n+2}$.

As all edge rewards are integer multiples of 0.25, also $\mathrm{Val}_\pi(u)$ is an integer multiple of 0.25 for every $u \in V_{\mathcal{B}_n}$ (starting in u, policy π visits every edge that has a non-zero reward either exactly once or never). This yields that $|\delta(\pi, v, w)|$ is a multiple of 0.25 as well, which concludes the proof. ◀

Note that, by Theorem 4 and as all vertex values w.r.t. π'_0 are zero, Dantzig$(\mathcal{D}_n, \pi'_0)$ only visits weak unichain policies with non-negative vertex values. Therefore, if we choose the probabilities $(p_v)_{v \in V_{\mathcal{B}_n} \setminus \{s\}}$ for increasing vertex numbers $\mathcal{N}_V(v)$ fast enough decreasing, then the previous lemma yields that Dantzig prefers improving switches that belong to vertices appearing early in the vertex numbering $\mathcal{N}_V$. We use this in the proof of Theorem 28.

The following technical result holds independently of the chosen pivot rule.

Algorithm 3 Dantzig(G, π).

input: Markov decision process G, weak unichain policy π for G

while π *admits an improving switch*:
  $\bar{s} \leftarrow$ an improving switch s for π maximizing $z_\pi(s)$
  $\pi \leftarrow \pi^{\bar{s}}$
return π

▶ **Lemma 25.** *Let $(v,w) \in E_{\mathcal{B}_n}$ be an improving switch that gets applied to a policy π during the execution of* Bland$(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$. *Let $\bar{\pi}'$ denote the policy for $\mathcal{D}_n$ that results from applying the switches $(x_{v,w}, y_{v,w})$ and $(v, x_{v,w})$ to the twin policy π' of π. Then, the edge $(x_{v,\pi(v)}, v)$ is improving for $\bar{\pi}'$ and it remains improving during the execution of* PolicyIteration$(\mathcal{D}_n, \bar{\pi}')$ *until it gets applied or until an improving switch of the form $(x_{u,\cdot}, y_{u,\cdot})$ with $\mathcal{N}_V(u) > \mathcal{N}_V(v)$ gets applied.*

With this, we can show that a certain class of pivot rules, including Bland's, Dantzig's, and the Largest Increase rule, yield an exponential number of improving switches on $\mathcal{D}_n$.

▶ **Lemma 26.** *Assume that* PolicyIteration$(\mathcal{D}_n, \pi'_0)$, *where π'_0 denotes the twin policy of π_0, gets applied with a pivot rule that satisfies the following conditions:*

(a) *For every improving switch $(v,w) \in E_{\mathcal{B}_n}$ that* Bland$(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$ *applies to some policy π,* PolicyIteration *applies $(x_{v,w}, y_{v,w})$ and $(v, x_{v,w})$ to the twin policy of π.*

(b) *While an edge of the form $(x_{v,\cdot}, v)$ is improving for some $v \in V_{\mathcal{B}_n}$,* PolicyIteration *does not apply an improving switch of the form $(x_{u,\cdot}, y_{u,\cdot})$ with $\mathcal{N}_V(u) > \mathcal{N}_V(v)$.*

Then, PolicyIteration$(\mathcal{D}_n, \pi'_0)$ *performs $\Omega(2^n)$ improving switches.*

Proof. Let π be a policy for $\mathcal{B}_n$ occurring during the execution of Bland$(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$, where we allow $\pi = \pi_0$, and let $(v,w) \in E_{\mathcal{B}_n}$ denote the switch that Bland applies to π. Let π' be the twin policy of π, and let $\tilde{\pi} = \pi^{(v,w)}$.

By condition (a), PolicyIteration applies $(x_{v,w}, y_{v,w})$ and $(v, x_{v,w})$ to π'. Denote the resulting policy by $\bar{\pi}'$.

According to Lemma 25, the edge $(x_{v,\pi(v)}, v)$ now stays improving until it gets applied as an improving switch or until an improving switch of the form $(x_{u,\cdot}, y_{u,\cdot})$ with $\mathcal{N}_V(u) > \mathcal{N}_V(v)$ gets applied. With condition (b), this yields that $(x_{v,\pi(v)}, v)$ gets applied by PolicyIteration at some point, and that it is constantly improving until then.

Note that, as long as $(v, x_{v,\pi(v)})$ is inactive, the policy's choice in $x_{v,\pi(v)}$ only affects the reduced costs of its unique incidental edge $(v, x_{v,\pi(v)})$. This edge is not active in $\bar{\pi}'$ and is not improving until the application of $(x_{v,\pi(v)}, v)$. Therefore, if we were to force the algorithm to apply $(x_{v,\pi(v)}, v)$ to $\bar{\pi}'$, this would not alter the remaining behavior of the algorithm. The policy resulting from this forced switch is the twin policy of $\tilde{\pi}$.

Hence, without changing the total number of applied improving switches (we only rearrange them), we can assume that PolicyIteration$(\mathcal{D}_n, \pi'_0)$ visits the twin policy of every policy visited by Bland$(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$. By Theorem 14, this yields that the algorithm needs to perform an exponential number of improving switches, which concludes the proof. ◀

Now it suffices to check the conditions given in Lemma 26 for each pivot rule.

▶ **Proposition 27.** *Let π'_0 denote the twin policy of π_0. Then,* Bland$(\mathcal{D}_n, \pi'_0, \mathcal{N}_{\mathcal{D}_n})$ *performs $\Omega(2^n)$ improving switches.*

Proof. We check the two conditions from Lemma 26. For condition (a), let π be a policy for $\mathcal{B}_n$ visited by Bland$(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$, including the case $\pi = \pi_0$, and let π' be the twin policy of π. Assume that Bland applies the improving switch $(v,w) \in E_{\mathcal{B}_n}$ to π.

By Observation 20, all improving switches for π' are of the form $(x_\cdot, y_\cdot)$. According to Lemma 21, the edge $(x_{v,w}, y_{v,w})$ is improving for π'. As (v,w) is the improving switch for π with the smallest Bland number in $\mathcal{N}_{\mathcal{B}_n}$, we know that, by construction of $\mathcal{N}_{\mathcal{D}_n}$, the algorithm applies the switch $(x_{v,w}, y_{v,w})$ to π'.

Further, since π is weak unichain due to Theorem 4, Lemma 22 yields that $(v, x_{v,w})$ is improving after this switch. As it is the successor of $(x_{v,w}, y_{v,w})$ in $\mathcal{N}_{\mathcal{D}_n}$ and as no other egde became improving due to the first switch, the algorithm applies $(v, x_{v,w})$ next. That is, Bland's rule satisfies condition (a).

Additionally, condition (b) holds since the edge $(x_{v,\pi(v)}, v)$ precedes any switch of the form $(x_{u,\cdot}, y_{u,\cdot})$ with $\mathcal{N}_V(u) > \mathcal{N}_V(v)$ in the Bland numbering $\mathcal{N}_{\mathcal{D}_n}$. ◀

As motivated above, the choice of the probabilities $(p_v)_{v \in V_{\mathcal{B}_n} \setminus \{s\}}$ in the following theorem yields that DANTZIG prefers improving switches that belong to vertices appearing early in the vertex numbering $\mathcal{N}_V$.

▶ **Theorem 28.** *Let $p_v = 2^{-\mathcal{N}_V(v)(n+5)}$ for all $v \in V_{\mathcal{B}_n} \setminus \{s\}$, and let π_0' denote the twin policy of π_0. Then, DANTZIG$(\mathcal{D}_n, \pi_0')$ performs $\Omega(2^n)$ improving switches.*

Proof. We check the two conditions from Lemma 26. For condition (b), we compute that the choice of the probabilities p_v yields that DANTZIG prefers improving switches belonging to vertices with a small vertex number.

Let $u, v \in V_{\mathcal{B}_n} \setminus \{s\}$ with $\mathcal{N}_V(u) > \mathcal{N}_V(v)$. Let further $e_u \in B(u)$ and $e_v \in B(v)$ be improving switches for some policy π for $\mathcal{D}_n$, which gets visited by POLICYITERATION$(\mathcal{D}_n, \pi_0')$. Then, π is weak unichain and only induces non-negative vertex values, so Lemma 24 yields

$$z_\pi(e_v) \geq p_v \cdot 0.25 = 2^{-\mathcal{N}_V(v)(n+5)-2} \geq 2^{-\mathcal{N}_V(u)(n+5)+(n+5)-2} = p_u \cdot 2^{(n+3)} > z_\pi(e_u).$$

Hence, Dantzig's rule prefers switches belonging to v over those belonging to u, so it satisfies condition (b).

For condition (a), let π be a policy for $\mathcal{B}_n$ visited by BLAND$(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$, including the case $\pi = \pi_0$, and let $(v, w) \in E_{\mathcal{B}_n}$ denote the switch that BLAND applies to π. Let π' be the twin policy of π.

By Observation 20, all improving switches for π' are of the form $(x_\cdot, y_\cdot)$. By construction of $\mathcal{N}_{\mathcal{D}_n}$, we know that BLAND prefers those switches $(x_\cdot, y_\cdot)$ that belong to vertices with a small vertex number. In the proof of Proposition 27, we see that BLAND$(\mathcal{D}_n, \pi', \mathcal{N}_{\mathcal{D}_n})$ applies $(x_{v,w}, y_{v,w})$ to π'. Since DANTZIG also prefers switches belonging to vertices with a small vertex number, we conclude that DANTZIG$(\mathcal{D}_n, \pi_0', \mathcal{N}_{\mathcal{D}_n})$ applies an improving switch to π' that belongs to v. However, there might be multiple such switches.

Recall that all improving switches for π' are of the form $(x_\cdot, y_\cdot)$. If $v = b_i$ for some $i \in [n]$, then only two of these (possibly improving) edges belong to v, one of which is active in π'. Therefore, in this case, DANTZIG applies the improving switch $(x_{v,w}, y_{v,w})$.

Now assume $v = a_i$ for some $i \in [n]$. Then, Observation 15 and Lemma 21 yield

$$z_{\pi'}(x_{a_i,b_i}, y_{a_i,b_i}) > \max\{z_{\pi'}(x_{a_i,a_{i+1}}, y_{a_i,a_{i+1}}), z_{\pi'}(x_{a_i,t}, y_{a_i,t})\}$$

if $(v, w) = \text{enter}(i)$, and

$$z_{\pi'}(x_{a_i,a_{i+1}}, y_{a_i,a_{i+1}}) > z_{\pi'}(x_{a_i,t}, y_{a_i,t})$$

if $(v, w) = \text{skip}(i)$. Therefore, the edge $(x_{v,w}, y_{v,w})$ has higher reduced costs than the other edges that belong to v, so DANTZIG applies it to π'.

Finally, if $v = t$, then Observation 16 and Lemma 21 yield that $(x_{v,w}, y_{v,w})$ is the only improving switch that belongs to t. Thus, it gets applied by DANTZIG.

We conclude that, in all cases, DANTZIG applies the switch $(x_{v,w}, y_{v,w})$ to π', which is the unique edge with the highest reduced costs. According to Lemma 22, the edge $(v, x_{v,w})$ has now the same reduced costs as $(x_{v,w}, y_{v,w})$ had before its application. Since $(v, x_{v,w})$ is the only edge that became improving during the last switch, DANTZIG applies this edge next. Therefore, Dantzig's rule also satisfies condition (a), which concludes the proof. ◀

Algorithm 4 LargestIncrease(G, π).

input: Markov decision process G, weak unichain policy π for G

while π *admits an improving switch*:
  $\bar{s} \leftarrow$ an improving switch s for π maximizing $\sum_{v \in V_n^A} \mathrm{Val}_{\pi^s}(v)$
  $\pi \leftarrow \pi^{\bar{s}}$
return π

Finally, we turn to policy iteration with the Largest Increase pivot rule, cf. Algorithm 4. In the most general sense, consider an arbitrary improving switch $s = (v, w)$ for some policy π. Assume that no ingoing edges of v are active in π. Then, the reduced costs of s coincide with the increase of the sum over all vertex values, that is, we obtain the equality $z_\pi(s) = \sum_{v \in V_n^A} \mathrm{Val}_{\pi^s}(v) - \sum_{v \in V_n^A} \mathrm{Val}_\pi(v)$. Further, the induced increase of the sum is always at least as large as the reduced costs.

From this, using the structure of $\mathcal{D}_n$, we can conclude that LargestIncrease mirrors the behavior of Dantzig if we choose the probabilities p_v as before.

▶ **Theorem 29.** *Let $p_v = 2^{-\mathcal{N}_V(v)(n+5)}$ for all $v \in V_{\mathcal{B}_n} \setminus \{s\}$, and let π_0' denote the twin policy of π_0. Then, LargestIncrease$(\mathcal{D}_n, \pi_0')$ performs $\Omega(2^n)$ improving switches.*

Proof. We check the two conditions from Lemma 26. For condition (a), let π be a policy for $\mathcal{B}_n$ occurring during the execution of Bland$(\mathcal{B}_n, \pi_0, \mathcal{N}_{\mathcal{B}_n})$, where we allow $\pi = \pi_0$, and let $(v, w) \in E_{\mathcal{B}_n}$ denote the switch that Bland applies to π. Let π' be the twin policy of π.

According to the proof of Theorem 28, Dantzig applies the improving switches $(x_{v,w}, y_{v,w})$ and $(v, x_{v,w})$ to π'. By Observation 20, all improving switches for π' are of the form $(x., y.)$, where π' does not reach $x.$ when starting in any other vertex. Therefore, since the probabilities $(p_v)_{v \in V_{\mathcal{B}_n} \setminus \{s\}}$ are chosen as in Theorem 28, the reduced costs of each of these improving switches coincide with the induced increase of the sum over all vertex values. Hence, LargestIncrease also applies the improving switch $(x_{v,w}, y_{v,w})$ to π'.

This switch only increases the reduced costs of the edge $(v, x_{v,w})$, which, by Lemma 22, coincide with the previous reduced costs. Therefore, the induced increase of the sum over all vertex values is for $(v, x_{v,w})$ now at least as large as it was for $(x_{v,w}, y_{v,w})$ before. Hence, LargestIncrease also applies $(v, x_{v,w})$ next. We conclude that the Largest Increase pivot rule satisfies condition (a).

Note that the reduced costs of the edges from condition (b) again coincide with the induced increase of the sum over all vertex values. Further, by the proof of Theorem 28, we know that Dantzig's rule prefers switches belonging to vertices with a small vertex number. Therefore, the Largest Increase rule also satisfies condition (b). ◀

Note that Theorem 1 is now a direct consequence of Theorems 27, 28, and 29. Moreover, we have seen in the proofs of these theorems that Bland's rule, Dantzig's rule, and the Largest Increase rule satisfy the conditions from Lemma 26. Thus, any combination of these rules also satisfies the conditions, which immediately yields Corollary 2. Finally, Corollary 3 follows from Theorem 5 together with Obervation 7, Lemma 8, and Observation 19.

References

1 Ilan Adler, Richard M Karp, and Ron Shamir. A simplex variant solving an m× d linear program in o (min (m2, d2) expected number of pivot steps. *Journal of Complexity*, 3(4):372–387, 1987.

2 Ilan Adler, Christos Papadimitriou, and Aviad Rubinstein. On simplex pivoting rules and complexity theory. In *Integer Programming and Combinatorial Optimization: 17th International Conference, IPCO 2014.*, pages 13–24. Springer, 2014.

3 Nina Amenta and Günter M Ziegler. Deformed products and maximal shadows of polytopes. *Contemporary Mathematics*, 223:57–90, 1999.

4 David Avis and Vasek Chvátal. Notes on Bland's pivoting rule. *Polyhedral Combinatorics: Dedicated to the memory of D.R. Fulkerson*, pages 24–34, 1978.

5 David Avis and Oliver Friedmann. An exponential lower bound for Cunningham's rule. *Mathematical Programming*, 161:271–305, 2017.

6 Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

7 Dimitris Bertsimas and Santosh Vempala. Solving convex programs by random walks. *Journal of the ACM*, 51(4):540–556, 2004.

8 Robert G Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2(2):103–107, 1977.

9 Karl-Heinz Borgwardt. The average number of pivot steps required by the simplex-method is polynomial. *Zeitschrift für Operations Research*, 26:157–177, 1982.

10 Daniel Dadush and Sophie Huiberts. A friendly smoothed analysis of the simplex method. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 390–403, 2018.

11 George Dantzig. *Linear programming and extensions*. Princeton university press, 1963.

12 George B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.

13 George B. Dantzig, Alex Orden, and Philip Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955.

14 Amit Deshpande and Daniel A Spielman. Improved smoothed analysis of the shadow vertex simplex method. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 349–356. IEEE, 2005.

15 Yann Disser, Oliver Friedmann, and Alexander V. Hopp. An exponential lower bound for zadeh's pivot rule. *Mathematical Programming*, 199(1-2):865–936, 2023.

16 Yann Disser and Martin Skutella. The simplex algorithm is NP-mighty. *ACM Transactions on Algorithms (TALG)*, 15(1):1–19, 2018.

17 John Dunagan and Santosh Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 315–320, 2004.

18 John Fearnley. Exponential lower bounds for policy iteration. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 551–562, 2010.

19 John Fearnley and Rahul Savani. The complexity of the simplex method. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 201–208, 2015.

20 Oliver Friedmann. *Exponential lower bounds for solving infinitary payoff games and linear programs*. PhD thesis, LMU Munich, 2011.

21 Oliver Friedmann, Thomas D. Hansen, and Uri Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 283–292, 2011.

22 Bernd Gärtner. The random-facet simplex algorithm on combinatorial cubes. *Random Structures & Algorithms*, 20(3):353–381, 2002.

23 Bernd Gärtner and Ingo Schurr. Linear programming and unique sink orientations. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 749–757, 2006.

24 Saul Gass and Thomas Saaty. The computational algorithm for the parametric objective function. *Naval research logistics quarterly*, 2(1-2):39–45, 1955.

25 Donald Goldfarb and William Y. Sit. Worst case behavior of the steepest edge simplex method. *Discrete Applied Mathematics*, 1(4):277–285, 1979.

26 Thomas D. Hansen. *Worst-case analysis of strategy iteration and the simplex method*. PhD thesis, Aarhus University, 2012.

27 Thomas D. Hansen and Uri Zwick. An improved version of the random-facet pivoting rule for the simplex algorithm. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–218, 2015.

28 Ronald A. Howard. *Dynamic programming and Markov processes*. John Wiley, 1960.

29 Sophie Huiberts, Yin Tat Lee, and Xinzhi Zhang. Upper and lower bounds on the smoothed complexity of the simplex method. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1904–1917, 2023.

30 Robert G. Jeroslow. The simplex algorithm with the pivot rule of maximizing criterion improvement. *Discrete Mathematics*, 4(4):367–377, 1973.

31 Gil Kalai. A subexponential randomized simplex algorithm. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 475–482, 1992.

32 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing (STOC)*, pages 302–311, 1984.

33 Jonathan A Kelner and Daniel A Spielman. A randomized polynomial-time simplex algorithm for linear programming. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 51–60, 2006.

34 Leonid G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

35 Victor Klee and George J Minty. How good is the simplex algorithm? *Inequalities*, 3(3):159–175, 1972.

36 Jiří Matoušek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996.

37 Mary Melekopoglou and Anne Condon. On the complexity of the policy improvement algorithm for Markov decision processes. *ORSA Journal on Computing*, 6(2):188–192, 1994.

38 Katta G. Murty. Computational complexity of parametric linear programming. *Mathematical Programming*, 19(1):213–219, 1980.

39 Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

40 Ingo Schurr and Tibor Szabó. Finding the sink takes some time: An almost quadratic lower bound for finding the sink of unique sink oriented cubes. *Discrete & Computational Geometry*, 31(4):627–642, 2004.

41 Steve Smale. Mathematical problems for the next century. *Mathematics: frontiers and perspectives*, pages 271–294, 2000.

42 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.

43 Tibor Szabó and Emo Welzl. Unique sink orientations of cubes. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 547–555, 2001.

44 Michael J Todd. Polynomial expected behavior of a pivoting algorithm for linear complementarity and linear programming problems. *Mathematical Programming*, 35(2):173–192, 1986.

45 Roman Vershynin. Beyond hirsch conjecture: walks on random polytopes and smoothed complexity of the simplex method. *SIAM Journal on Computing*, 39(2):646–678, 2009.

Exact Matching: Correct Parity and FPT Parameterized by Independence Number

Nicolas El Maalouly
Department of Computer Science, ETH Zürich, Switzerland

Raphael Steiner
Department of Computer Science, ETH Zürich, Switzerland

Lasse Wulf
Institute of Discrete Mathematics, TU Graz, Austria

Abstract

Given an integer k and a graph where every edge is colored either red or blue, the goal of the exact matching problem is to find a perfect matching with the property that exactly k of its edges are red. Soon after Papadimitriou and Yannakakis (JACM 1982) introduced the problem, a randomized polynomial-time algorithm solving the problem was described by Mulmuley et al. (Combinatorica 1987). Despite a lot of effort, it is still not known today whether a deterministic polynomial-time algorithm exists. This makes the exact matching problem an important candidate to test the popular conjecture that the complexity classes P and RP are equal. In a recent article (MFCS 2022), progress was made towards this goal by showing that for bipartite graphs of bounded bipartite independence number, a polynomial time algorithm exists. In terms of parameterized complexity, this algorithm was an XP-algorithm parameterized by the bipartite independence number. In this article, we introduce novel algorithmic techniques that allow us to obtain an FPT-algorithm. If the input is a general graph we show that one can at least compute a perfect matching M which has the correct number of red edges modulo 2, in polynomial time. This is motivated by our last result, in which we prove that an FPT algorithm for general graphs, parameterized by the independence number, reduces to the problem of finding in polynomial time a perfect matching M with at most k red edges and the correct number of red edges modulo 2.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Perfect Matching, Exact Matching, Independence Number, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.28

Related Version *Full Version*: https://arxiv.org/abs/2207.09797

Funding *Raphael Steiner*: Supported by an ETH Zurich Postdoctoral Fellowship.
Lasse Wulf: Supported by the Austrian Science Fund (FWF): W1230.

1 Introduction

In the *Exact Matching Problem* (denoted from now on by EM), we are given a graph G together with a fixed coloring of its edges in two colors (red and blue). The question is, for a given integer k, to decide whether there exists a perfect matching M of G with the additional property that exactly k of the edges of the perfect matching M are red. Clearly, if we have the special case that all edges of the graph are red and $k = |V(G)|/2$ then this problem is simply to decide whether there exists a perfect matching in the graph, which is well-known to be decidable in polynomial time [7]. However, when the coloring of the edges is heterogeneous, the problem difficulty seems to increase significantly (see below).

Papadimitriou and Yannakakis [28] initially introduced EM in 1982 and conjectured it to be NP-hard. However, a randomized polynomial-time algorithm solving the problem was described by Mulmuley, Vazirani and Vazirani in 1987 in the course of their celebrated isolation lemma [26]. Given standard complexity theoretic hypotheses, this makes it unlikely for EM to be NP-hard. Despite the existence of a polynomial-time randomized algorithm, as of today it is still not known whether EM can also be solved in deterministic polynomial time. The algorithm of Mulmuley et al. uses polynomial identity testing and is based on the Schwartz-Zippel Lemma [29, 35], which has resisted all attempts of derandomization so far. Indeed, EM is one of the few natural problems which has a randomized polynomial-time algorithm (i.e. it is contained in the complexity class RP) but for which it is not known whether it admits a deterministic polynomial-time algorithm (i.e. it is contained in P). It is a major open conjecture that RP=P, and so EM becomes a natural candidate to test this hypothesis.

For this reason, EM has been cited in several papers as an open problem. This includes recent breakthrough papers such as the seminal work on the parallel computation complexity of the matching problem [31], works on planarizing gadgets for perfect matchings [17], works on budgeted, color bounded, or constrained matching problems [3, 22, 24, 25, 30], on multicriteria optimization problems [16] and on matroid intersection problems [5]. It is further known that several different problems relate directly or indirectly to EM. The following is a non-exhaustive list of examples: EM is polynomial-time equivalent to the DNA sequencing problem [4]. EM is equivalent to a variant of the problem of finding a solution of a binary linear equation system with small Hamming weight [2]. EM can be reduced to a special case of the recoverable robust assignment problem [12].

Previous work. Progress in finding deterministic algorithms for EM (and therefore finding positive evidence for the conjecture P=RP) has only been made for restricted graph classes: It is known that EM can be solved in determinisitic polynomial time for planar and more generally $K_{3,3}$-minor free graphs [34], as well as graphs of bounded genus [13]. These works use Pfaffian orientations to derandomize the algebraic technique from [26]. EM can also be solved for graphs of bounded treewidth using a dynamic programming approach [8, 32]. In contrast to these classes of sparse graphs, EM on dense graphs seems to be even harder: Already solving the problem on complete graphs and complete bipartite graphs is highly nontrivial. In fact, at least 4 articles just dealing with this special case have appeared in the literature [14, 18, 21, 33]. Recent work [9] made a step forward by showing how to solve EM on graphs of constant independence number, where the *independence number* of a graph G is defined as the largest number α such that G contains an *independent set* of size α, and bipartite graphs of constant bipartite independence number, where the *bipartite independence number* of a bipartite graph G equipped with a bipartition of its vertices is defined as the largest number β such that G contains a *balanced independent set* of size 2β, i.e., an independent set using exactly β vertices from both color classes. This generalizes previous results for complete and complete bipartite graphs which correspond to the special cases $\alpha = 1$ and $\beta = 0$. The authors presented an XP-algorithm, i.e. an algorithm running in time $O(n^{f(\alpha)})$, for the problem. The existence of an FPT algorithm, i.e. an algorithm with running time $f(\alpha)n^{O(1)}$, was left as an open question. The authors also conjectured that counting perfect matching is #P-hard for this class of graphs. This conjecture was later proven in [11] already for $\alpha = 2$ or $\beta = 3$. As a consequence, the Pfaffian derandomization technique is unlikely to work for this class of graphs, because this technique implicitly counts the number of perfect matchings. This makes the graph class of graphs of bounded

independence number a promising frontier to push the limits of deterministic techniques. To the best of our knowledge, these are the only results from the last 40 years showing that EM can be solved in deterministic poly-time for restricted graph classes.

Apart from restricted graph classes, one can also consider parameterized algorithms for EM, using the natural parameter k. Note that an XP-algorithm in this case is trivial to obtain using brute-force guessing (guess the red edges that go in a solution and complete the perfect matching using only blue edges). An FPT algorithm would, however, be highly desirable as it is likely provide a lot of insight into EM. The only progress towards that goal can be found in [8] where some color coding tools were developed but only applied to the almost trivial case of bounded circumference graphs.

Another direction of progress towards solving EM is the study of relaxed versions of it. A first such relaxation would be to lift the requirement for a perfect matching. In [34], however, it was shown that there is a simple deterministic polynomial time algorithm such that given a "Yes" instance of EM, computes an *almost* perfect matching (i.e. of size at least $\frac{n}{2} - 1$) containing k red edges. This result is as close to optimal as possible for this type of relaxation. The study of the other type of relaxation, i.e. relaxing the color constraints, was only recently initiated in [8]. It was shown that there is a deterministic polynomial time algorithm which given a "Yes"-instance of EM outputs a perfect matching with k' red edges, such that $k/2 \leq k' \leq 3k/2$.

Exact matchings modulo 2. A crucial tool in this paper is to consider matchings with k' red edges, where $k' \equiv_2 k$, that is, matchings of the correct parity. Let $r(M)$ denote the number of red edges in a matching M. We define the *Correct Parity Matching Problem* (CPM), where given a red-blue edge-colored graph and an integer k, the goal is to find a perfect matching M such that $r(M) \equiv_2 k$. Note that parity problems (and more general congruency-constrained problems) have been studied in the context of other graph algorithms [19, 27], but are not well studied for the perfect matching problem. A more challenging version of the problem, *Bounded Correct Parity Matching* (BCPM), requires finding a perfect matching M such that $r(M) \equiv_2 k$ *and* $r(M) \leq k$. In [20] the complexity of the EM problem was even further highlighted by showing that the Exact Matching polytope has exponential extension complexity even when restricted to the bipartite case and to the parity constraint (i.e. CPM in bipartite graphs has exponential extension complexity).

Our results. From now on, when we say that an algorithm has *polynomial running time*, we mean a deterministic algorithm, whose running time is bounded by $\text{poly}(n)$, even if α, β, k are not bounded by a constant. We show in this paper how BCPM can be used to solve EM. Precisely, our results are the following:

- We show that EM reduces to BCPM in FPT time parameterized by α in the following sense: There exists an algorithm, which performs a single oracle call to BCPM, and solves EM on general graphs, in running time $f(\alpha)n^{O(1)}$. (The result holds analogously for the bipartite independence number β). Without access to the BCPM oracle, the algorithm outputs a perfect matching with either $k-1$ or k red edges or deduces that the answer of the given EM-instance is "No" (Section 3).
- CPM can be solved in polynomial time for all graphs (Section 4). This insight is based on a deep result by Lovász [23].
- On bipartite graphs, the more difficult problem BCPM can be solved in polynomial time (Theorem 17). As a consequence, there is an FPT algorithm parameterized by β which solves EM on bipartite graphs (Theorem 2).

Due to space restrictions, proofs of statements marked ⋆ can be found in the extended version of this paper [10].

2 Preliminaries

All graphs considered are simple. For $G = (V, E)$, we let $V(G) := V$ and $E(G) := E$. We always use the letter n to denote the number of vertices of a graph G, i.e. $n = |V(G)|$. An *edge-colored* graph is a tuple (G, col), where $\text{col} : E \to \{\text{red}, \text{blue}\}$ prescribes a color to each edge. An instance of EM is a tuple (G, col, k). Given an instance of EM and a perfect matching (abbreviated PM) M, we define edge weights $w_M : E \to \mathbb{N}$ as follows: We have $w_M(e) = 0$ if e is a blue edge, $w_M(e) = +1$ if e is a red non-matching edge and $w_M(e) = -1$ if e is a red matching edge. The weight function w_M plays a critical role in many arguments in this paper. When the PM M can be deduced from context, we may write w instead of w_M. In this case, the *weight* of edge e is $w_M(e)$. For G' a subgraph of G, we define $R(G')$ (resp. $B(G')$) to be the set of red (resp. blue) edges in G', $r(G') := |R(G')|$ to be the number of red edges of G' and $w_M(G')$ to be the sum of the weights of edges in G'. If $\mathcal{C}$ is a set of vertex-disjoint cycles, then we define $w_M(\mathcal{C}) = \sum_{C \in \mathcal{C}} w_M(C)$.

We say that a set of disjoint cycles or paths is M-alternating if for any two adjacent edges in the set, one of them is in M and the other is not. Undirected cycles are considered to have an arbitrary orientation. For a cycle C and $u, v \in C$, $C[u, v]$ is defined as the path from u to v along C (in the fixed but arbitrarily chosen orientation). The term $\text{Ram}(r, s)$ refers to the Ramsey number, i.e. every graph on $\text{Ram}(r, s)$ vertices contains either a clique of size r or an independent set of size s. For simplicity we will use the following upper bound: $\text{Ram}(s + 1, s + 1) < 4^s$ [15].

3 Reducing EM to BCPM in FPT time

The goal of this section is to prove our two main theorems:

▶ **Theorem 1.** *EM can be reduced to BCPM in FPT time parameterized by the independence number of the graph.*

▶ **Theorem 2.** *There exists an FPT algorithm for EM on bipartite graphs parameterized by the bipartite independence number of the graph.*

We will first introduce the algorithm and then prove Theorem 1 in Section 3.4 and Theorem 2 in Section 3.5. Finally, in Section 3.6 we will discuss the case where a BCPM oracle can not be used.

3.1 Tools from Prior Work

The algorithm we develop to prove Theorems 1 and 2 will rely on many of the tools developed in [9] and [8]. We start with the two main propositions that we aim to use. The setting of both propositions is the same: We are given some PM M explicitly, and we know that there is another PM M' which we know exists, but we do not know explicitly. We are given the PM M and the number $r(M')$ as input and would like to find either M' itself, or at least another PM M'' with $r(M'') = r(M')$.

▶ **Proposition 3** (from [9])**.** *Let M and M' be two PMs in G such that $|B(M \Delta M')| \leq L$ or $|R(M \Delta M')| \leq L$, for $L \geq 1$. Then there exists a deterministic algorithm running in time $n^{O(L)}$ such that given M and $r(M')$, it outputs a PM M'' with $r(M'') = r(M')$.*

▶ **Proposition 4** (adapted from [8]). *(⋆) Given a graph $G = (V, E)$ with edge colors red and blue, let M and M' be two PMs in G such that $|E(M\Delta M')| \leq L$, for $L \geq 1$. Then there exists an algorithm running in time $f(L)n^{O(1)}$ (for $f(L) = L^{O(L)}$) such that given M and $r(M')$, it outputs a PM M'' with $r(M'') = r(M')$.*

The algorithm from Proposition 4 is faster (FPT instead of XP when parameterized by L), but it requires more assumptions on M'. The algorithm from Proposition 3 works by guessing which L edges are in $R(M\Delta M')$ (respectively $B(M\Delta M')$) and then checks if the red (blue) edges can be completed to a PM by using only blue (red) edges. The algorithm from Proposition 4 works using color-coding technique (see [6, Chapter 5] for more details on color coding).

In [9] the authors show that for graphs of small independence number, one could use Proposition 3 to get an XP algorithm (parameterized by the independence number) by bounding either the number of red edges or the number of blue edges in the symmetric difference with a target matching M'. Our aim is to show that we can use the stronger Proposition 4 from [8] to get an FPT algorithm, which would require that we bound both color classes (i.e. the entire symmetric difference). This turns out to be much more difficult to achieve and requires novel algorithmic techniques that we describe in the next section. Our algorithm does, however, start by bounding one of the color classes before bounding the second. For that we simply rely on the tools developed in [9] to avoid starting from scratch. Due to the technicality of some of the used tools, some readers might want to skip the details of the tools from previous work and jump ahead to the next section, only coming back to these definitions and lemmas when needed.

A crucial concept to understand the tools from prior work is a property of the weight function $w = w_M$ as defined in Section 2. Let M and M' be two perfect matchings. It is well-known that the symmetric difference $\mathcal{C} := M\Delta M'$ is a set of edges that forms a vertex-disjoint union of M-alternating cycles. An easy observation is now that $w_M(\mathcal{C})$ counts the difference of red edges between M and M', that is, we have $r(M') = r(M) + w_M(\mathcal{C})$. This follows directly from the definition of w_M. The second crucial concept is the concept of a *skip*.

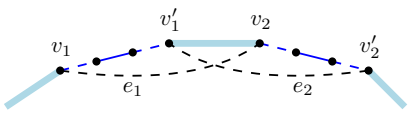


Figure 1 A skip formed by two non-matching edges e_1 and e_2 (in black). Matching edges are normal lines, non-matching edges are dashed. The bold lines represent subpaths.

▶ **Definition 5** (from [9]). *Let M be a PM and C an M-alternating cycle. A skip S is a set of two non-matching edges $e_1 := (v_1, v_2)$ and $e_2 := (v_1', v_2')$ with $e_1, e_2 \notin C$ and $v_1, v_1', v_2, v_2' \in C$ (appearing in this order along C) such that $C' = e_1 \cup e_2 \cup C \setminus (C[v_1, v_1'] \cup C[v_2, v_2'])$ is an M-alternating cycle, $|C| - |C'| > 0$ and $|w_M(S)| \leq 4$ where $w_M(S) := w_M(C') - w_M(C)$ is called the weight of the skip.*

Let M, C, S, C' be as above. We say that *using the skip* S is the action of replacing the alternating cycle C by the alternating cycle C'. If furthermore M' is another PM and $C \in M\Delta M'$, then we say that using S also modifies M' the following way: We let $M'_{\text{new}} = M'\Delta C\Delta C' = M\Delta(((M\Delta M') \setminus C) \cup C')$. In other words, M'_{new} is the matching which has the same symmetric difference from M as M', except that C was replaced by C'. It is an easy observation that M'_{new} is again a PM and $r(M'_{\text{new}}) = r(M') + w(S)$. This means

that using a positive skip (i.e. a skip of strictly positive weight) increases the cycle weight, using a negative skip decreases it and using a *0-skip* (i.e. a skip of weight 0) does not change the cycle weight. Using a skip always results in a cycle of smaller cardinality. If $P \subseteq C$ is a path and $C[v_1, v_2'] \subseteq P$, then we say that P contains the skip S. Two skips $\{(v_1, v_2), (v_1', v_2')\}$ and $\{(u_1, u_2), (u_1', u_2')\}$ are called disjoint if they are contained in disjoint paths along the cycle. Note that two disjoint skips can be used independently. Finally, observe that iterating over all skips of a given alternating cycle C can be done in polynomial time by trying all possible combinations of two chords from the cycle C and checking whether they form a skip. This means that if a skip with certain properties is shown to exist, it can also be found in polynomial time.

▶ **Definition 6** (from [9]). *Let M be a PM and $\mathcal{C}$ a set of disjoint M-alternating cycles. A 0-skip set with respect to $\mathcal{C}$ is a set of disjoint skips on cycles of $\mathcal{C}$ such that the total weight of the skips is 0.*

▶ **Definition 7** (from [9]). *Let M be a PM and $\mathcal{C}$ a set of disjoint M-alternating cycles. A 0-skip-cycle set with respect to $\mathcal{C}$ is a set of disjoint skips on cycles of $\mathcal{C}$ and/or cycles from $\mathcal{C}$, such that the total weight of the skips minus the total weight of the cycles is 0.*

We say that *using a skip-cycle set* $\mathcal{S}$ means to change $\mathcal{C}$ by removing all cycles in $\mathcal{S}$ from $\mathcal{C}$ and by using all skips in $\mathcal{S}$ (i.e. for every $S \in \mathcal{S}$ that is a skip, locate the corresponding cycle $C \in \mathcal{C}$ with S in C and use S on C). A perfect matching M'_{new} is defined in an analogous way as M'_{new} is defined for using a single skip (i.e., such that $M'_{\text{new}} = M \Delta \mathcal{C}_{\text{new}}$). If $\mathcal{S}$ was a 0-skip-cycle set, then we have $r(M'_{\text{new}}) = r(M')$. Using a 0-skip-cycle set always decreases the total size of $\mathcal{C}$. In this paper, it will be a common strategy to locate 0-skip-cycle sets contained in the symmetric difference $M \Delta M'$ of two PMs. If we manage to find such a 0-skip-cycle set, then using it on $M \Delta M'$ will produce a new PM M'_{new} such that $r(M'_{\text{new}}) = r(M')$, but $|M \Delta M'_{\text{new}}| < |M \Delta M'|$. Hence we make progress in the sense that we reduce the symmetric difference $M \Delta M'$ while maintaining $r(M')$.

The following lemmas are taken and adapted from [9]. They show that under certain assumptions 0-skip sets or 0-skip-cycle sets always exist. They are adapted to also include a proof that the desired objects can be found in polynomial time. We leave their adapted proofs to the appendix.

▶ **Lemma 8** (adapted from [9]). *($\star$) Let M be a PM and P an M-alternating path with $w_M(P) \geq 2t \cdot 4^\alpha$ (resp. $w_M(P) \leq -2t \cdot 4^\alpha$), for $t \geq 1$, then P contains at least t disjoint negative (resp. positive) skips. If P and M are given, then we can also find t such skips in polynomial time.*

▶ **Lemma 9** (adapted from [9]). *($\star$) Let $t \geq 8 \cdot 4^\alpha$ and $t' = 4t^2$. Let M be a PM and $\mathcal{C}$ a set of disjoint M-alternating cycles and $C \in \mathcal{C}$ such that $|w_M(\mathcal{C})| \leq t'$ and $|w_M(C)| \geq 2t'$, then $\mathcal{C}$ contains a 0-skip-cycle set. If $\mathcal{C}, M$ are given, we can also find a 0-skip-cycle set in polynomial time.*

▶ **Lemma 10** (adapted from [9]). *($\star$) Let $t \geq 3$. Let M be a PM and $\mathcal{C}$ a set of disjoint M-alternating cycles such that $|w_M(\mathcal{C})| \leq t$, $|w_M(C)| \leq 2t$ for all $C \in \mathcal{C}$ and $|\mathcal{C}| \geq 10t^3$, then $\mathcal{C}$ contains a 0-skip-cycle set. If $\mathcal{C}, M$ are given, we can also find a 0-skip-cycle set in polynomial time.*

▶ **Lemma 11** (adapted from [9]). *($\star$) Let $t \geq 8 \cdot 4^\alpha$. Let M be a PM and $\mathcal{C}$ a set of disjoint M-alternating cycles such that $|\mathcal{C}| \leq 10t^3$, $|w_M(C)| \leq 2t$ for all $C \in \mathcal{C}$ and $\mathcal{C}$ contains at least $1000t^6$ blue edges and $1000t^6$ red edges, then $\mathcal{C}$ contains a 0-skip set. If $\mathcal{C}, M$ are given, then we can also find a 0-skip set in polynomial time.*

3.2 The Main Algorithm

The aim of this section is to present the algorithm which reduces EM to BCPM in time $f(\alpha)n^{O(1)}$. We first sketch the idea of the algorithm: We assume that the algorithm receives two PMs M and M' as input, such that $r(M) < k < r(M')$ and such that both M and M' already have the correct parity, i.e. $r(M) \equiv_2 r(M') \equiv_2 k$. We will later show how this can be done with an oracle call to BCPM. But even in the case where the BCPM oracle can not be used and M, M' are just some arbitrary PMs with $r(M) < k < r(M')$, our algorithm still computes something meaningful: We show that in this case a PM with k or $k-1$ red edges will be output, or it will be deduced that the given EM-instance has answer "No". This variant of the algorithm is further discussed in Section 3.6.

Our algorithm modifies the PMs M and M' many times. But the invariant is maintained that during the whole execution of the algorithm, both the PMs M and M' will never change their parity. The basic idea of the algorithm is to have many iterations, where in each iteration either M is modified such that $r(M)$ increases by 2, or M' is modified such that $r(M')$ decreases by two. Clearly, if we can do such a modification in every iteration, we will eventually arrive at a PM with k red edges. One might ask why we consider modifications of the kind $+2$ and -2, instead of the kind $+1$ and -1. The reason for this is that a change of ± 1 might not always be possible, even in complete graphs. To see this, consider the smallest possible modification of a PM. It consists in taking its symmetric difference with an alternating cycle of length four. Such a cycle may add or remove up to two red edges from the matching and it is possible that we only find such cycles adding or removing exactly two red edges. On the converse, if all small cycles add or remove one red edge, we can still achieve a change of two by simply considering two such cycles.

However, reality is more complicated and even a ± 2 modification might not always be possible. The first hurdle is that such a modification might not be possible if $r(M) \ll r(M')$. To combat this hurdle, the algorithm splits into three phases, where in the first phase the PMs M, M' are modified such that they keep their parity and after their modification we have that $r(M), r(M')$ are close to k. Details for phase 1 will be provided in Lemma 12. In the second phase, we will do many iterations, such that in each iteration the algorithm tries to (i) increase $r(M)$ by 2, or (ii) decrease $r(M')$ by 2, or (iii) strictly decrease the cardinality of the symmetric difference $|E(M \Delta M')|$. Finally, it can still happen that neither (i), (ii), or (iii) are possible. However, we prove a key lemma which states that in this situation (and if the given EM-instance is a "Yes" instance), we can use color coding techniques to find a PM M^* in time $f(\alpha)n^{O(1)}$ which is a solution to EM, i.e. $r(M^*) = k$. The algorithm then enters phase 3, where it either finds M^* or deduces that the given EM-instance is a "No"-instance. We now provide the formal description of the algorithm:

Input: A red-blue edge-colored graph (G, col), a nonnegative integer k. Two PMs M and M' with $r(M) < k < r(M')$.

Phase 1: Find two PMs M_{new} and M'_{new} such that

$$k - 8 \cdot 4^{\alpha} \leq r(M_{\text{new}}) \leq k \leq r(M'_{\text{new}}) \leq k + 8 \cdot 4^{\alpha},$$

and such that the parity is maintained, i.e. $r(M_{\text{new}}) \equiv_2 r(M)$ and $r(M'_{\text{new}}) \equiv_2 r(M')$. Set $M \leftarrow M_{\text{new}}$ and $M' \leftarrow M'_{\text{new}}$. If this step fails, output "EM-instance has no solution".

Phase 2: If either M or M' is a solution matching we are done. Otherwise repeat the following three steps until either M or M' is a solution matching or until every step (i),(ii), and (iii) fails in the same iteration:

(i) Invoke the algorithm of Proposition 3 with respect to the matching M and $L = 2$ in order to try to find a PM M_{new} with $r(M_{\text{new}}) = r(M) + 2$. If such a PM is found, let $M \leftarrow M_{\text{new}}$, otherwise do not modify M and consider step (i) as failed.

(ii) Invoke the algorithm of Proposition 3 with respect to the matching M' and $L = 2$ in order to try to find a PM M'_{new} with $r(M'_{\text{new}}) = r(M') - 2$. If such a PM is found, let $M' \leftarrow M'_{\text{new}}$, otherwise do not modify M' and consider step (ii) as failed.

(iii) Invoke the algorithms of Lemma 9, Lemma 10 or Lemma 11 (with $t = 256 \cdot 4^{2\alpha}$), to try to find a 0-skip or a 0-skip-cycle set in $M\Delta M'$. If such an object is found, then use it (i.e. change M' accordingly) to reduce $|E(M\Delta M')|$. Otherwise do not modify M, M' and consider step (iii) as failed.

Phase 3: If either M or M' is a solution matching we are done. Otherwise invoke the algorithm of Proposition 4 with $L = 2^{\alpha^{O(1)}}$ (for appropriately large constants) on the matching M to try to find a PM M^* with $r(M^*) = k$. If such a PM M^* is found, then output it. Otherwise output "EM-instance has no solution".

This completes the description of the algorithm. The remainder of this section is dedicated to its proof. First, we prove that phase 1 can be completed correctly in polynomial time (Lemma 12). It is not so difficult to prove that phase 2 requires only polynomial time (as there are at most n^2 iterations). Finally, we prove in our main lemma (Lemma 15) that if steps (i),(ii),(iii) all fail simultaneously, then phase 3 is guaranteed to succeed. This is the most difficult lemma to prove. In Section 3.4 we summarize the proof and explain how to obtain the two initial matchings M and M' required as input for phase 1.

Finally, we describe the modifications necessary for bipartite graphs (Section 3.5) and for cases where the BCPM oracle is not available (Section 3.6).

3.3 Proof of the main lemmas

The following lemmas help us prove the correctness and polynomial running time of the algorithm.

▶ **Lemma 12.** *Given a "Yes" instance of EM and two PMs M and M' with $r(M) \leq k \leq r(M')$, there exists a deterministic polynomial time algorithm that outputs two PMs M_1 and M_2 with $r(M_1) \equiv_2 r(M)$, $r(M_2) \equiv_2 r(M')$ and $k - 8 \cdot 4^\alpha \leq r(M_1) \leq k \leq r(M_2) \leq k + 8 \cdot 4^\alpha$.*

Proof. As long as $r(M) < k - 8 \cdot 4^\alpha$ we will consider two cases:

- All cycles $C \in M\Delta M'$ have weight $w_M(C) \leq 4 \cdot 4^\alpha$. In this case $M\Delta M'$ must contain at least two strictly positive cycles C_1 and C_2. If $w_M(C_1) \equiv_2 0$ then we replace M by $M\Delta C_1$ and iterate (note that $r(M) < r(M\Delta C_1) \leq k$ and $r(M\Delta C_1) \equiv_2 r(M)$). The case $w_M(C_2) \equiv_2 0$ is similar. Otherwise we replace M by $M\Delta(C \cup C')$ and iterate (note that $r(M) < r(M\Delta(C \cup C')) \leq k$ and $r(M\Delta(C \cup C')) \equiv_2 r(M)$).
- There exists $C \in M\Delta M'$ with $w_M(C) > 4 \cdot 4^\alpha$. Observe that $C \in M'\Delta M$ and $w_{M'}(C) = -w_M(C) \leq -4 \cdot 4^\alpha$. By Lemma 8 applied to M' and $w_{M'}$, we have that C contains two positive skips (with respect to M' and $w_{M'}$). If any of the skips has even weight, we use it to increase the weight of $w_{M'}(C)$ and iterate (note that $r(M)$ increases since using a skip in $M'\Delta M$ modifies M). Otherwise we use both skips. In either case, $r(M)$ increases and its parity is preserved. Note that $r(M)$ can increase by at most 8 given that a skip must have weight at most 4 by definition.

In both cases $r(M)$ increases after every iteration. So there can be at most $O(n)$ iterations, each running in polynomial time, until $k - 8 \cdot 4^\alpha \leq r(M) \leq k$. Now we apply a similar procedure to decrease $r(M')$. As long as $r(M') > k + 8 \cdot 4^\alpha$ we will consider two cases:

- All cycles in $M'\Delta M$ have weight $w_{M'}$ more than $-4 \cdot 4^\alpha$. In this case $M'\Delta M$ must contain at least two strictly negative cycles C_1 and C_2. If $w_{M'}(C_1) \equiv_2 0$ then we replace M' by $M'\Delta C_1$ and iterate (note that $k \leq r(M'\Delta C_1) < r(M')$ and $r(M'\Delta C_1) \equiv_2 r(M')$). The case $w_{M'}(C_2) \equiv_2 0$ is similar. Otherwise we replace M' by $M'\Delta(C \cup C')$ and iterate (note that $k \leq r(M\Delta(C \cup C')) < r(M')$ and $r(M\Delta(C \cup C')) \equiv_2 r(M')$).
- There exists $C \in M'\Delta M$ with $w_{M'}(C) < -4 \cdot 4^\alpha$. Observe that $C \in M\Delta M'$ with $w_M(C) = -w_{M'}(C) \geq 4 \cdot 4^\alpha$. By Lemma 8 applied to M and w_M, C contains two negative skips (with respect to M and w_M). If any of the skips has even weight, we use it to reduce $w_M(C)$ and iterate (note that $r(M')$ decreases since using a skip in $M\Delta M'$ modifies M'). Otherwise we use both skips. In either case, $r(M')$ decreases and its parity is preserved. Note that $r(M')$ can decrease by at most 8 given that a skip must have weight at least -4 by definition.

In both cases $r(M')$ decreases after every iteration. So there can be at most $O(n)$ iterations, each running in polynomial time, until $k \leq r(M') \leq k + 8 \cdot 4^\alpha$. Finally the algorithm terminates by outputting $M_1 := M$ and $M_2 := M'$. ◀

▶ **Lemma 13.** *Let M be a PM and $\mathcal{C}$ a set of disjoint M-alternating cycles with the following properties:*

- *$\mathcal{C}$ does not contain monochromatic cycles.*
- *$|E(\mathcal{C})| \geq 2t^3$.*
- *$|R(\mathcal{C})| \leq t$ (resp. $|B(\mathcal{C})| \leq t$).*

Then $\mathcal{C}$ contains a blue (resp. red) M-alternating path of length at least t.

Proof. We will consider the case when $|R(\mathcal{C})| \leq t$. The case $|B(\mathcal{C})| \leq t$ is proven similarly by swapping the two colors. First observe that if $\mathcal{C}$ contains at most t red edges and no monochromatic cycles, then $|\mathcal{C}| \leq t$. So by the pigeonhole principle, $\mathcal{C}$ must contain a cycle C with $|E(C)| \geq 2t^2$. Consider the set of maximal blue subpaths of C and let p_B be the number of these paths. As every such path is accompanied by a red edge, we have $p_B \leq t$. Finally, C has at least $2t^2 - t$ blue edges, so by the pigeonhole principle one of the blue paths must have length at least $(2t^2 - t)/t \geq t$. ◀

The above lemma simply shows that if only one color class is bounded, there must be long monochromatic paths of the other color. The next lemma shows that the existence of long monochromatic paths in turn implies the existence of small cycles.

▶ **Lemma 14.** *Let M be a PM and C an M-alternating cycle. Let $P \subseteq C$ be a blue (resp. red) M-alternating path of length at least $6\,\mathrm{Ram}(\mathrm{Ram}(4, \alpha+1), \alpha+1)$, starting with a non-matching edge and not containing 0-skips. Then there must be two edges $e_1 := (b_1, b_2)$ and $e_2 := (w_1, w_2)$ with endpoints on P, at least one of which must be red (resp. blue), such that $C' = e_1 \cup e_2 \cup C[b_1, w_1] \cup C[b_2, w_2]$ is an M-alternating cycle with $0 < w_M(C') \leq 2$ (resp. $-2 \leq w_M(C') < 0$) and containing a number of red (resp. blue) edges equal to the absolute value of its weight.*

Proof. We will only deal with the case when P is blue, the other case is treated similarly (by switching the roles of the two colors in the proof). We assume that P has an arbitrary orientation which is used to define the start and end vertices of subpaths of P. First, we divide P into a set of consecutive paths $\mathcal{P}$ of length 6 each, starting with the first non-matching edge. Let $\mathcal{P}_1$ be the set of paths formed by the first 3 edges of each path in $\mathcal{P}$. The set of start vertices of paths in $\mathcal{P}_1$ has size at least $\mathrm{Ram}(\mathrm{Ram}(4, \alpha+1), \alpha+1)$ so it must contain a clique Q of size $\mathrm{Ram}(4, \alpha+1)$. Let $\mathcal{P}_2$ be the set of paths in $\mathcal{P}_1$ with start vertices in Q. The set of end vertices of paths in $\mathcal{P}_2$ must contain a clique Q' of size 4 (see Figure 2). Let

$\mathcal{P}_3 := \{P_1, P_2, P_3, P_4\}$ be the set of paths in $\mathcal{P}_1$ with end vertices in Q'. Let s_i and t_i be the start and end vertices of path P_i for $i \in \{1, 2, 3, 4\}$. Observe that any two distinct paths $P_i, P_j \in \mathcal{P}_3$ have their endpoints connected by the edges (s_i, s_j) and (t_i, t_j) and a skip is created this way. If both edges were blue, we would get a 0-skip (since the whole path P has only blue edges). Letting $i = 2, j = 3$, we see that one of the edges (s_2, s_3) or (t_2, t_3) must be red. Suppose (s_2, s_3) is red. Observe that $(t_1, t_2) \cup (s_2, s_3) \cup C[t_1, s_2] \cup C[t_2, s_3]$ is a cycle of weight $+1$ or $+2$ (depending on whether (t_1, t_2) is red or blue, since all other edges are blue) and containing at most 2 red edges. Similarly, suppose (t_2, t_3) is red. Observe that $(t_2, t_3) \cup (s_3, s_4) \cup C[t_2, s_3] \cup C[t_3, s_4]$ is a cycle of weight $+1$ or $+2$ (depending on whether (s_3, s_4) is red or blue) and the number of red (resp. blue) edges it contains is equal to the absolute value of its weight. ◀

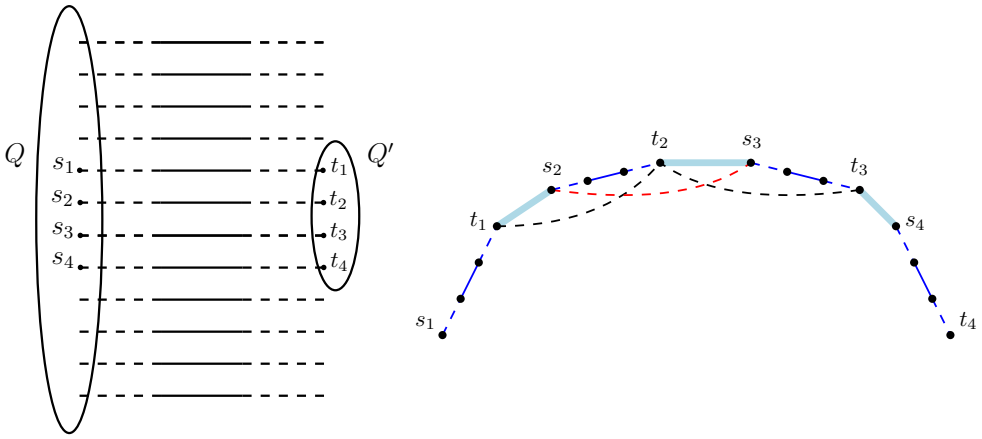


Figure 2 Left: The set of paths $\mathcal{P}_2$, of size $\text{Ram}(4, \alpha + 1)$, and the cliques Q and Q'. Right: The paths from $\mathcal{P}_3$ along the blue path P and the vertices s_i, t_j. Matching edges are normal lines, non-matching edges are dashed. The bold lines between t_i and s_{i+1} represent subpaths. Observe that $\{(s_2, s_3), (t_2, t_3)\}$ forms a skip and $(t_1, t_2) \cup (s_2, s_3) \cup C[t_1, s_2] \cup C[t_2, s_3]$ is an alternating cycle.

Finally, we are ready to prove our main lemma. Roughly speaking, it states that if all steps (i),(ii) and (iii) in phase two of the algorithm fail, then phase 3 is guaranteed to succeed. More specifically, it states that if we cannot make small progress towards a solution then we are ready to apply Proposition 4 and find one in FPT time. Small progress here means either getting the number of red edges in M or M' closer to k, or making their symmetric difference smaller.

▶ **Lemma 15.** *Let M and M' be two PMs with the following properties:*

(a) $r(M) < k - 1$, $r(M') > k$.

(b) $|w_M(M \Delta M')| \leq t$ *for* $t = 256 \cdot 4^{2\alpha}$.

(c) *There is no PM M_1 such that $r(M_1) = r(M) + 2$ and $|R(M \Delta M_1)| = 2$.*

(d) *There is no PM M'_1 such that $r(M'_1) = r(M') - 2$ and $|B(M' \Delta M'_1)| = 2$.*

(e) *$M \Delta M'$ does not contain any 0-skip.*

(f) *The algorithms of Lemma 9, Lemma 10 and Lemma 11 all fail to find a 0-skip-cycle set in $M \Delta M'$.*

If there is at least one PM with k red edges, then there exists a PM M^* such that $r(M^*) = k$ and $|E(M \Delta M^*)| = 2^{\alpha^{O(1)}}$.

Proof. We start by giving a high level overview and the intuition behind the proof. First we note that properties (a) and (b) will be guaranteed after phase 1 of the algorithm and they state that both M and M' are close to k in terms of number of red edges. Second, properties (c) and (d) state that our algorithm is unable to make small progress in terms of getting the number of red edges in M or M' closer to k. Finally properties (e) and (f) state that our algorithm is unable to make small progress in term of making the symmetric difference between M and M' smaller.

Our final goal is to bound the symmetric difference between M and some solution M^*. We will do that by contradiction to one of the given properties. Observe that if the conditions for Lemma 14 are met, i.e. there is a long blue only M-alternating path, then the lemma guarantees that small progress towards getting the number of red edges closer to k is possible. The same holds for long red only M'-alternating paths. Note, however, that we might need to apply the lemma twice in order to ensure that the progress is in increments or decrements of two, thus contradicting either property (c) or (d). This way we can bound the length of blue only M-alternating paths. Then if red only M-alternating paths are also bounded, Lemma 13 implies that either both colors are bounded, in which case we are done, or none of them is. In the latter case, we use the machinery developed in [9] (see Section 3.1) to reach the contradiction (remember that the goal there was to bound one color class), and this requires properties (a) and (b) to hold. The same holds if blue only M'-alternating paths are bounded.

The only remaining obstacles are long red only M-alternating or blue only M'-alternating paths. To deal with that, we try to reduce the symmetric difference between M and M' such that long monochromatic M-alternating paths are also M'-alternating (and the contradiction above can again be reached) since the two matchings do not differ by that many edges. Bounding the symmetric difference between M and M' relies on a contradiction to properties (e) or (f). It follows the same steps as bounding the symmetric difference between M and M^*, but with the added benefit that paths in this symmetric difference are both M and M'-alternating, which avoids the problem of long red only M-alternating or blue only M'-alternating paths.

To summarise, we start by bounding $|E(M\Delta M')|$ (first bounding one color class, then the second). Then we are able to bound $|E(M\Delta M^*)|$ (again one color class at a time).

Detailed proof. We will start by showing that one color class of $M\Delta M'$ must be bounded. This allows us to then bound $|E(M\Delta M')|$. We then consider the solution matching M^* that minimizes $|E(M\Delta M^*)|$ and start by bounding the number of blue edges in $M\Delta M^*$. Finally, we also show that the number of red edges in $M\Delta M^*$ is bounded, thus bounding $|E(M\Delta M^*)|$.

Bounding one color class of $M\Delta M'$. Since we failed to reduce $|E(M\Delta M')|$ using the algorithm of Lemma 9, the weight of all cycles in $M\Delta M'$ must be bounded: $|w(C)| \leq 2t$ for all $C \in M\Delta M'$. Since we failed to reduce $|E(M\Delta M')|$ using the algorithm of Lemma 10, the number of cycles in $M\Delta M'$ must be bounded: $|M\Delta M'| \leq 10t^3$. Finally, since we failed to reduce $|E(M\Delta M')|$ using the algorithm of Lemma 11, $|B(M\Delta M')|$ or $|R(M\Delta M')|$ must be bounded (by $1000t^6$). Let $t' = \max(1000t^6, 20\,\mathrm{Ram}(\mathrm{Ram}(4, \alpha+1), \alpha+1))$ (note that $1000t^6 = 2^{O(\alpha)}$, $\mathrm{Ram}(4, \alpha+1) = \alpha^{O(1)}$ and $\mathrm{Ram}(\mathrm{Ram}(4, \alpha+1), \alpha+1)) = 2^{\alpha^{O(1)}}$ so $t' = 2^{\alpha^{O(1)}}$).

Bounding $|E(M\Delta M')|$. First, we show that property (c) implies that there is no blue M-alternating path of length at least t' in the graph. Suppose such a path exists. Divide the path into two blue paths P_1 and P_2 of length at least $t'/2$ each. From Lemma 14 applied to each of the paths P_1 and P_2, we get that there exists two disjoint M-alternating cycles C_1 and C_2 with $0 < w_M(C_1) \leq 2$, $0 < w_M(C_2) \leq 2$ and each containing a number of red edges equal to the absolute value of their weight. If C_1 contains two red edges, let $M_1 := M\Delta C_1$. Otherwise if C_2 contains two red edges, let $M_1 := M\Delta C_2$. Finally, if both C_1 and C_2 contain only one red edge, let $M_1 := M\Delta(C_1 \cup C_2)$. Observe that $|R(M\Delta M_1)| = 2$ and $r(M_1) = r(M) + 2$, contradicting property (c).

Next we show that property (d) implies that there is no red M'-alternating path of length at least t' in the graph. Divide the path into two red paths P_1 and P_2 of length at least $t'/2$ each. From Lemma 14 applied to each of the paths P_1 and P_2, we get that there exists two disjoint M-alternating (with respect to M') cycles C_1 and C_2 with $-2 \leq w_{M'}(C_1) < 0$, $-2 \leq w_{M'}(C_2) < 0$ and and each containing a number of blue edges equal to the absolute value of their weight. If C_1 contains two blue edges, let $M'_1 := M'\Delta C_1$. Otherwise if C_2 contains two blue edges, let $M'_1 := M'\Delta C_2$. Finally, if both C_1 and C_2 contain only one blue edge, let $M'_1 := M'\Delta(C_1 \cup C_2)$. Observe that $|B(M'\Delta M'_1)| = 2$ and $r(M'_1) = r(M') - 2$, contradicting property (d).

Suppose $|R(M\Delta M')| \geq 2t'^3$. Then by the previous paragraph $|B(M\Delta M')| \leq t'$. Note that $M\Delta M'$ contains no monochromatic cyle, as this would be a 0-skip cycle set, therefore by Lemma 13, $M'\Delta M$ contains a red M'-alternating path of length at least t'. But this contradicts property (c). Now suppose $|B(M\Delta M')| \geq 2t'^3$. Then $|R(M\Delta M')| \leq t'$ and by Lemma 13, $M\Delta M'$ contains a blue M-alternating path of length at least t', contradicting property (c). So we get $|E(M\Delta M')| = |B(M\Delta M')| + |B(M\Delta M')| \leq 4t'^3$.

Bounding $|B(M\Delta M^*)|$. Now let M^* among all those PMs with k red edges be the one which minimizes $|E(M\Delta M^*)|$. Note that $|w_M(M\Delta M^*)| \leq |w_M(M\Delta M')| \leq t$. Since $|E(M\Delta M^*)|$ is minimal, $M\Delta M^*$ cannot contain a 0-skip-cycle set. By Lemma 9, the weight of all cycles in $M\Delta M^*$ must be bounded: $|w(C)| \leq 2t$ for all $C \in M\Delta M^*$. By Lemma 10, the number of cycles in $M\Delta M^*$ must be bounded: $|M\Delta M^*| \leq 10t^3$. Finally, by Lemma 11, $|B(M\Delta M^*)|$ or $|R(M\Delta M^*)|$ must be bounded (by $1000t^6 \leq t'$). Suppose $|B(M\Delta M^*)| > 2t'^3$. So $|R(M\Delta M^*)| \leq t'$ and by Lemma 13, $M\Delta M^*$ contains a blue M-alternating path of length t', contradicting property (c). So $|B(M\Delta M^*)| \leq 2t'^3$.

Bounding $|E(M\Delta M^*)|$. Let $t'' = 4t'^4$. Suppose $|R(M\Delta M^*)| > 2t''^3$, by Lemma 13 $M^*\Delta M$ contains a red path P with $|P| \geq t''$. Observe that $M'\Delta M^* = (M\Delta M^*)\Delta(M\Delta M')$ and $P \subseteq M\Delta M^*$ so

$$P \cap (M'\Delta M^*) = P \backslash (P \cap (M\Delta M')).$$

We have $|P \cap (M\Delta M')| \leq |E(M\Delta M')| < 4t'^3$, so if all paths in $P \cap (M'\Delta M^*)$ have length at most t' then $|P| < 4t'^4$, a contradiction. So there must be a path $P' \subseteq P \cap (M'\Delta M^*)$ of length at least t'. Note that P' is a red M'-alternating path, contradicting property (d). So we have $|E(M\Delta M^*)| \leq 4t''^3 = t''^{O(1)} = t'^{O(1)} = 2^{\alpha^{O(1)}}$. ◀

3.4 Main theorem for general graphs

Proof of Theorem 1. Suppose we have a polynomial time oracle for BCPM. We start by solving CPM on the given instance. This can be done in polynomial time (as we prove later in Theorem 19) and will give us a PM M_p with $r(M_p) \equiv_2 k$. If $r(M_p) \geq k$, let $M' := M_p$

and use an oracle call to BCPM to get M with $r(M) \equiv_2 k$ and $r(M) \leq k$. Otherwise, if $r(M_p) \geq k$, let $M := M_p$ and use an oracle call to BCPM to get M' with $r(M') \equiv_2 k$ and $r(M') \geq k$ (this can simply be done by swapping the red and blue colors and using $k' = n/2 - k$ as parameter for the BCPM oracle). In both cases, we obtain PMs M, M' such that $r(M) \equiv_2 k \equiv_2 r(M')$ and $r(M) \leq k \leq r(M')$.

Note that if this step fails (in the sense that the CPM or BCPM call returns "false"), then the EM instance has no solution. Otherwise we apply the algorithm of Section 3.2 on the EM-instance with M and M' as input. Our goal now is to prove that if the EM-instance is a "YES" instance, then the following must be true:

(a) Phase 1 runs in polynomial time and outputs two PMs M and M' such that $k - 8 \cdot 4^\alpha \leq r(M) \leq k \leq r(M') \leq k + 8 \cdot 4^\alpha$, $r(M) \equiv_2 r(M') \equiv_2 k$.

(b) Phase 2 runs in polynomial time and either outputs a PM with k red edges (and the algorithm terminates) or a PM M such that there exists a PM M^* with $r(M^*) = k$ and $|E(M \Delta M^*)| \leq 2^{\alpha^{O(1)}}$ (for appropriately large constants).

(c) If the algorithm did not terminate in Phase 2, then Phase 3 runs in time $f(\alpha)n^{O(1)}$ and outputs a PM with k red edges.

It is easy to see that if all the above items hold, then the algorithm runs in time $f(\alpha)n^{O(1)}$ and always outputs a PM with k red edges if one exists. Note that (a) and (c) follow directly from Lemma 12 and Proposition 4 respectively.

To prove (b) first observe that as long as $r(M) \neq k$ and $r(M') \neq k$, all steps in phase 2 maintain the following invariants: $r(M) \leq k \leq r(M')$ and $r(M) \equiv_2 r(M') \equiv_2 k$. To see this, note that $r(M)$ and $r(M')$ can only change by 2 every step and they start with the same parity as k. So in order for $r(M)$ to go above k or $r(M')$ to go below k they would need to pass by k, at which point the algorithm terminates. Also observe that if any of the steps does not fail, then either $r(M') - r(M)$ decreases or $|E(M \Delta M')|$ decreases while $r(M') - r(M)$ remains unchanged. So if we consider as a measure of progress $r(M') - r(M)$ and $|E(M \Delta M')|$ ordered lexicographically (where progress is towards smaller values of the measure), then we always make progress (i.e. the measure strictly decreases). Note that $r(M') - r(M) \leq n$ and is always non-negative and the same holds for $|E(M \Delta M')|$. So the algorithm can perform at most n^2 iterations in phase 2. Since every iteration runs in polynomial time (this is true for steps (i) and (ii) by Proposition 3 and for step (iii) by Lemma 9, Lemma 10 and Lemma 11), we get that phase 2 runs in polynomial time. Now observe that the algorithm only terminates in phase 2 if either M or M' is a solution (i.e. it has k red edges). So it remains to show that if the algorithm does not terminate in this phase then there exists a PM M^* with $r(M^*) = k$ and $|E(M \Delta M^*)| \leq 2^{\alpha^{O(1)}}$. Observe that in case of non-termination, all the conditions of Lemma 15 are met:

(a) $r(M) < k - 1$, $r(M') > k$: follows from the invariants and M, M' not being solutions.

(b) $|w_M(M \Delta M')| \leq 256 \cdot 4^{2\alpha}$: follows from $r(M') - r(M) \leq 16 \cdot 4^\alpha$.

(c) There is no PM M_1 such that $r(M_1) = r(M) + 2$ and $|R(M \Delta M_1)| = 2$: follows from the failure of (i).

(d) There is no PM M'_1 such that $r(M'_1) = r(M') - 2$ and $|B(M' \Delta M'_1)| = 2$: follows from the failure of (ii).

(e) $M \Delta M'$ does not contain any 0-skip: follows from the failure of (iii).

(f) The algorithms of Lemma 9, Lemma 10 and Lemma 11 all fail to find a 0-skip-cycle set in $M \Delta M'$: follows from the failure of (iii).

So by Lemma 15 we get the desired result. ◀

3.5 Main theorem for bipartite graphs

In order to prove the main theorem for the bipartite case (Theorem 2), we start by proving a similar result to the main theorem on general graph that is adapted to bipartite graphs, i.e., we use the bipartite independence number of the graph (the proof can be found in the extended version [10]).

▶ **Lemma 16.** ($\star$) *EM on bipartite graphs can be reduced to BCPM on bipartite graphs in FPT time parameterized by the bipartite independence number of the graph.*

It remains to show that there is a deterministic polynomial time algorithm for BCPM on bipartite graphs. This result can be derived from the more general result of [1] on network matrices, as noted in [20], even for the more general weighted version of the problem. To make it more accessible, we reprove it using a standard dynamic programming techniques. The high level approach, as briefly described in [20], is the following: start by computing a minimum weight perfect matching, in our case a perfect matching with minimum number of red edges, and if the number of red edges is even then find a minimum odd weight alternating cycle and output the symmetric difference. We could not find formal proof of correctness and running time for this algorithm in the literature, therefore we provide one in the extended version of this paper [10].

▶ **Theorem 17.** ($\star$) *There is a deterministic polynomial time algorithm for BCPM on bipartite graphs.*

3.6 Main theorem without oracle access

Although an FPT algorithm parameterized by the independence number for general graphs still requires an algorithm for BCPM, the following theorem shows that without relying on BCPM the algorithm developed in this section can still output a PM that is very close to optimal, i.e. it contains either k or $k-1$ red edges (the proof is similar to that of Theorem 1 and left for the extended version [10]).

▶ **Theorem 18.** ($\star$) *There exists an algorithm such that given a "Yes" instance of EM, it outputs a perfect matching with either $k-1$ or k red edges in time $f(\alpha)n^{O(1)}$.*

This strengthens the results of [8] by reducing the constraint violation to at most one red edge at the expense of an FPT (parameterized by the independence number) instead of a polynomial running time.

4 Correct Parity Matching for General Graphs

While solving BCPM for general graphs remains an open problem, in this section we present a solution to the easier problem of CPM which is only concerned with the parity of the number of red edges.

▶ **Theorem 19.** ($\star$) *There is a deterministic polynomial time algorithm for CPM.*

We will establish Theorem 19 as a consequence of a deep result by Lovász [23] on the *linear hull* of perfect matchings of a graph. We first need to introduce some notation which we adopt from [23]. Let a (not necessarily bipartite) graph $G = (V, E)$ and a field $\mathbb{F}$ be given, and let us denote by $\mathcal{M}$ the set of perfect matchings of G. Then the linear hull of perfect matchings $\mathrm{lin}_{\mathbb{F}}(\mathcal{M})$ is the linear subspace of $\mathbb{F}^E$, generated by the characteristic vectors of

perfect matchings in G. Concretely, $\text{lin}_{\mathbb{F}}(\mathcal{M})$ is the linear span of $\{1_M | M \in \mathcal{M}\}$, where for every perfect matching M the vector $1_M \in \mathbb{F}^E$ is defined by $1_M(e) = 1$ for every $e \in M$ and $1_M(e) = 0$ for every $e \in E \setminus M$.

We will make use of the following result of Lovász [23].

▶ **Theorem 20** ([23]). *For every finite field $\mathbb{F}$ there is a deterministic polynomial-time algorithm that, given as input a graph G, returns a linear basis of $lin_{\mathbb{F}}(\mathcal{M})$.*

The importance of this result by Lovász for solving the CPM is explained through the following lemma. As usual, for two vectors $x, y \in \mathbb{F}^E$ we denote by $\langle x, y\rangle := \sum_{e\in E} x_e y_e \in \mathbb{F}$ their scalar product.

▶ **Lemma 21.** *Let $G = (V, E)$ be a graph equipped with a coloring of its edges with colors red and blue. Let $\mathbb{F}_2$ denote the 2-element field and let $\{x_1, \ldots, x_d\} \subset \mathbb{F}_2^E$ be a linear basis of $lin_{\mathbb{F}_2}(\mathcal{M})$. Let $r \in \mathbb{F}_2^E$ be defined by $r_e := 1$ for all red edges $e \in E$ and $r_e = 0$ for all blue edges $e \in E$. Then the following two statements are equivalent:*

1. *There exists a perfect matching M in G containing an odd number of red edges.*
2. *There exists $i \in \{1, \ldots, d\}$ such that $\langle x_i, r\rangle = 1$.*

Proof. Suppose first that there exists a perfect matching M in G containing an odd number of red edges. Then the incidence vector $1_M \in \text{lin}_{\mathbb{F}_2}(\mathcal{M})$ can be represented as a linear combination of the basis elements $x_1, \ldots, x_d$, in other words, there exists $I \subseteq \{1, \ldots, d\}$ such that

$$1_M = \sum_{i\in I} x_i.$$

Taking scalar products with r we get

$$\langle 1_M, r\rangle = \sum_{i\in I} \langle x_i, r\rangle.$$

Note that the scalar product on the left hand side equals the number of red edges in M taken modulo 2, and hence it equals 1. But then at least one of the scalar products on the right hand side must also be non-zero, i.e., there exists $i \in I$ with $\langle x_i, r\rangle = 1$, as desired.

Conversely, suppose there exists $i \in \{1, \ldots, d\}$ such that $\langle x_i, r\rangle = 1$. Then by virtue of $\text{lin}_{\mathbb{F}_2}(\mathcal{M})$ being spanned by the characteristic vectors of perfect matchings in M, there exists a list of perfect matchings $M_1, \ldots, M_t$ in G such that $x_i = \sum_{j=1}^{t} 1_{M_j}$. Using the same argument as above, i.e., by taking scalar products with r and using that $\langle x_i, r\rangle = 1$, we find that there must exist $j \in \{1, \ldots, t\}$ with $\langle 1_{M_j}, r\rangle = 1$, which means that M_j is a perfect matching of G with an odd number of red edges. This concludes the proof. ◀

We may now deduce the following.

▶ **Corollary 22.** *There exists a deterministic polynomial-time algorithm, that, given as input a red-blue edge-colored graph $G = (V, E)$ and a number $k \in \mathbb{Z}$, decides whether or not G contains a perfect matching M with $r(M) \equiv_2 k$.*

Proof. Suppose first that k is odd. We use Theorem 20 to compute in deterministic polynomial time a linear basis $x_1, \ldots, x_d$ of $\text{lin}_{\mathbb{F}_2}(\mathcal{M})$. Note that since $\text{lin}_{\mathbb{F}_2}(\mathcal{M})$ is a subspace of $\mathbb{F}_2^E$, its dimension satisfies $d \leq |E|$. Next we generate the incidence vector r of red edges as in the previous lemma, and compute the scalar products $\langle x_i, r\rangle$ for $i = 1, \ldots, d$ in polynomial time. If at least one of these products equals 1, we return that a perfect matching M with $r(M) \equiv_2 k$ exists, and otherwise we return that such a matching does not exist. The correctness of this output follows by Lemma 21.

Next suppose k is even. Let G' be the red, blue-edge colored graph which is obtained as the disjoint union of G with its given edge-coloring and a disjoint new edge of color red. The perfect matchings of G' are exactly the perfect matchings of G together with the additional new red edge, and hence G contains a perfect matching M with $r(M) \equiv_2 k$ if and only if G' contains a perfect matching with and odd number of red edges. Thus we can decide whether such a matching exists by invoking the algorithm from the case $k = 1$ described above with G' as the input. ◀

It is now easy to use the above decision-version of the CPM to solve the CPM itself by a standard edge-deletion procedure.

Proof of Theorem 19. Let $G = (V, E)$ be the input graph with a given red, blue-edge coloring, and let further $k \in \mathbb{Z}$ be given. We use Corollary 22 to decide if G contains a perfect matching M with $r(M) \equiv_2 k$. If it does not, then the algorithm stops with this conclusion. Otherwise, we search through the edges $e \in E$ one by one, and for each such edge test (again using Corollary 22) whether $G - e$ contains a perfect matching M with $r(M) \equiv_2 k$.

Suppose first we find an edge $e \in E$ such that $G - e$ contains a perfect matching M with $r(M) \equiv_2 k$. In this case we make a recursive call of the algorithm to $G - e$, which will return a perfect matching with the correct parity in $G - e$. We can then return this matching, as it is also a perfect matching in G with the correct parity of red edges.

Otherwise, we find that there exists no $e \in E$ such that $G - e$ contains a perfect matching M with $r(M) \equiv_2 k$. But as we know that G does contain a perfect matching M with $r(M) \equiv_2 k$, this means that all edges of G are contained in M, and hence we may return the set of edges of G and thereby find a solution to the CPM. ◀

5 Conclusion and Open Problems

So far, EM has only been solved for very sparse graphs (i.e. bounded tree-width and bounded genus graphs) and very dense graphs (i.e. bounded independence number graphs). The techniques used are quite different between these two cases. Especially in the case of dense graphs, many previous works considered only complete (bipartite) graphs without much progress. Only recently, the results were extended to the case of bounded independence number, leading to XP algorithms parameterized by α or β. Looking for FPT algorithms was the natural next step. In this paper, we could resolve the bipartite case fully, while the non-bipartite case could only be resolved partially. However, our results in the non-bipartite case still yield the following two non-trivial, independent insights: (i) To obtain an FPT algorithm parameterized by α it suffices to solve BCPM (Theorem 1) and the easier problem CPM can always be solved (Theorem 19). (ii) An FPT algorithm parameterized by α can w.l.o.g. assume to start with a PM with $k - 1$ red edges (Theorem 18).

We hope that these insights can be the starting point for future work to obtain an FPT algorithm parameterized by α, or, even better, an FPT algorithm parameterized by k. The latter would be considered quite a breakthrough as it is likely to require a lot of deep understanding of the structure and patterns behind EM, given the difficulty of making progress towards it.

References

1 Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A strongly polynomial algorithm for bimodular integer linear programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1206–1219, 2017.

2 Vikraman Arvind, Johannes Köbler, Sebastian Kuhnert, and Jacobo Torán. Solving linear equations parameterized by hamming weight. *Algorithmica*, 75(2):322–338, 2016.

3 André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128(1):355–372, 2011.

4 Jacek Błażewicz, Piotr Formanowicz, Marta Kasprzak, Petra Schuurman, and Gerhard J Woeginger. A polynomial time equivalence between DNA sequencing and the exact perfect matching problem. *Discrete Optimization*, 4(2):154–162, 2007.

5 Paolo M. Camerini, Giulia Galbiati, and Francesco Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2):258–273, 1992.

6 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

7 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

8 Nicolas El Maalouly. Exact matching: Algorithms and related problems. *arXiv preprint*, 2022. `arXiv:2203.13899`.

9 Nicolas El Maalouly and Raphael Steiner. Exact Matching in Graphs of Bounded Independence Number. In *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:14, 2022.

10 Nicolas El Maalouly, Raphael Steiner, and Lasse Wulf. Exact matching: Correct parity and FPT parameterized by independence number. *CoRR*, abs/2207.09797, 2022. `doi:10.48550/arXiv.2207.09797`.

11 Nicolas El Maalouly and Yanheng Wang. Counting perfect matchings in dense graphs is hard. *arXiv preprint*, 2022. `arXiv:2210.15014`.

12 Dennis Fischer, Tim A Hartmann, Stefan Lendl, and Gerhard J Woeginger. An investigation of the recoverable robust assignment problem. *arXiv preprint*, 2020. `arXiv:2010.11456`.

13 Anna Galluccio and Martin Loebl. On the theory of Pfaffian orientations. I. Perfect matchings and permanents. *Electronic Journal of Combinatorics*, 6:R6, 1999.

14 Hans-Florian Geerdes and Jácint Szabó. A unified proof for Karzanov's exact matching theorem. Technical Report QP-2011-02, Egerváry Research Group, Budapest, 2011.

15 Ronald L Graham, Bruce L Rothschild, and Joel H Spencer. *Ramsey theory*, volume 20. John Wiley & Sons, 1991.

16 Fabrizio Grandoni and Rico Zenklusen. Optimization with more than one budget. *arXiv preprint*, 2010. `arXiv:1002.2147`.

17 Rohit Gurjar, Arpita Korwar, Jochen Messner, Simon Straub, and Thomas Thierauf. Planarizing gadgets for perfect matching do not exist. In *International Symposium on Mathematical Foundations of Computer Science*, pages 478–490. Springer, 2012.

18 Rohit Gurjar, Arpita Korwar, Jochen Messner, and Thomas Thierauf. Exact perfect matching in complete graphs. *ACM Transactions on Computation Theory (TOCT)*, 9(2):1–20, 2017.

19 Edith Hemaspaandra, Holger Spakowski, and Mayur Thakur. Complexity of cycle length modularity problems in graphs. In *Latin American Symposium on Theoretical Informatics*, pages 509–518. Springer, 2004.

20 Xinrui Jia, Ola Svensson, and Weiqiang Yuan. The exact bipartite matching polytope has exponential extension complexity. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1635–1654. SIAM, 2023.

21 AV Karzanov. Maximum matching of given weight in complete and complete bipartite graphs. *Cybernetics*, 23(1):8–13, 1987.

22 Steven Kelk and Georgios Stamoulis. Integrality gaps for colorful matchings. *Discrete Optimization*, 32:73–92, 2019.

23 László Lovász. Matching structure and the matching lattice. *Journal of Combinatorial Theory, Series B*, 43:187–222, 1987.

24 Monaldo Mastrolilli and Georgios Stamoulis. Constrained matching problems in bipartite graphs. In *International Symposium on Combinatorial Optimization*, pages 344–355. Springer, 2012.

25 Monaldo Mastrolilli and Georgios Stamoulis. Bi-criteria and approximation algorithms for restricted matchings. *Theoretical Computer Science*, 540:115–132, 2014.

26 Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

27 Martin Nägele, Benny Sudakov, and Rico Zenklusen. Submodular minimization under congruency constraints. *Combinatorica*, 39(6):1351–1386, 2019.

28 Christos H Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM (JACM)*, 29(2):285–309, 1982.

29 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

30 Georgios Stamoulis. Approximation algorithms for bounded color matchings via convex decompositions. In *International Symposium on Mathematical Foundations of Computer Science*, pages 625–636. Springer, 2014.

31 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-NC. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 696–707. Ieee, 2017.

32 Moshe Y Vardi and Zhiwei Zhang. Quantum-inspired perfect matching under vertex-color constraints. *arXiv preprint*, 2022. `arXiv:2209.13063`.

33 Tongnyoul Yi, Katta G Murty, and Cosimo Spera. Matchings in colored bipartite networks. *Discrete Applied Mathematics*, 121(1-3):261–277, 2002.

34 Raphael Yuster. Almost exact matchings. *Algorithmica*, 63(1):39–50, 2012.

35 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International Symposium on Symbolic and Algebraic Computation (EUROSAM 1979)*, pages 216–226. Springer, 1979.

Approximation Guarantees for Shortest Superstrings: Simpler and Better

Matthias Englert
University of Warwick, Coventry, UK

Nicolaos Matsakis
Charles University, Prague, Czech Republic

Pavel Veselý
Charles University, Prague, Czech Republic

Abstract

The Shortest Superstring problem is an NP-hard problem, in which given as input a set of strings, we are looking for a string of minimum length that contains all input strings as substrings. The Greedy Conjecture (Tarhio and Ukkonen, 1988) states that the GREEDY algorithm, which repeatedly merges the two strings of maximum overlap, is 2-approximate. We have recently shown (STOC 2022) that the approximation guarantee of GREEDY is at most $\frac{13+\sqrt{57}}{6} \approx 3.425$. Before that, the best established upper bound for this was 3.5 by Kaplan and Shafrir (IPL 2005), which improved upon the upper bound of 4 by Blum et al. (STOC 1991). To derive our previous result, we established two incomparable upper bounds on the overlap sum of all cycle-closing edges in an optimal cycle cover and utilized lemmas of Blum et al.

We improve the more involved one of the two bounds and, at the same time, make its proof more straightforward. This results in an improved approximation guarantee of $\frac{\sqrt{67}+2}{3} \approx 3.396$ for GREEDY. Additionally, our result implies an algorithm for the Shortest Superstring problem having an approximation guarantee of $\frac{\sqrt{67}+14}{9} \approx 2.466$, improving slightly upon the previously best guarantee of $\frac{\sqrt{57}+37}{18} \approx 2.475$ (STOC 2022).

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Shortest Superstring problem, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.29

Funding *Nicolaos Matsakis*: Supported by GA ČR project 22-22997S.
Pavel Veselý: Partially supported by GA ČR project 22-22997S and by Center for Foundations of Modern Computer Science (Charles University project UNCE/SCI/004).

1 Introduction

The shortest superstring problem naturally models a scenario when we have a set of overlapping strings which we need to represent in a compressed form. However, unlike in typical lossless data compression such as Lempel-Ziv schemes, we would like the input strings to be human-readable in the result. That is, the compressed representation of input strings should be a string over the same alphabet that contains all of the strings as substrings. This viewpoint of superstrings as compressed representations has been the crux of their very recent application for representing k-mers, which are k-long substrings of a genomic sequence [19]. These k-mers are typically highly overlapping and in such cases, the shortest superstring of k-mers has length close to the theoretical minimum of the number of distinct k-mers.

Formally, we define the Shortest Superstring problem (SSP) as follows: For a given set of strings S (over a fixed alphabet), compute a minimum-length common *superstring* for the input strings, i.e., a string that contains any $s \in S$ as a substring. SSP is a classical and well-studied problem mentioned in several algorithmic textbooks, e.g., [25, 18, 9, 5]. SSP

is APX-hard (i.e., it is NP-hard to obtain a $(1+\varepsilon)$-approximation for some $\varepsilon > 0$) and remains so even when restricted to binary alphabets or input strings having the same length $r \geq 3$ [24].

Therefore, assuming P $\neq$ NP, the best we can hope for are constant-guarantee approximation algorithms. However, determining the best possible constant guarantee is a long-standing open problem, studied for more than three decades. First, Blum et al. [3] designed an algorithm for which they proved an upper bound of 3 on its approximation ratio. Several papers subsequently obtained better approximations using various algorithms [22, 6, 13, 1, 2, 4, 20, 16] and the currently best approximation guarantee is $\frac{37+\sqrt{57}}{18} \approx 2.475$ [7]. In contrast, the hardness result only rules out a 1.003-approximation [12].

Perhaps the most well-known approximation algorithm for SSP is GREEDY which iteratively merges two strings of maximum overlap until only one string remains (if there are more pairs of strings with maximum overlap, we choose arbitrarily). GREEDY is an appealing choice to implement in practice due to its simplicity and close-to-optimal results in experiments [8, 14, 19]. However, the worst-case behavior of GREEDY is far from understood. Blum et al. [3] showed that GREEDY is 4-approximate, an upper bound which was improved to 3.5 by Kaplan and Shafrir [11] and recently, in our previous work, to $\frac{13+\sqrt{57}}{6} \approx 3.425$ [7]. It is easy to see that GREEDY is at least 2-approximate by considering the input $\{c(ab)^k, (ba)^k, (ab)^k c\}$ for $k \to \infty$ [21]. The *Greedy Conjecture* states that this lower bound is tight [21]. Despite an extensive effort to prove or disprove this, the three works [3, 11, 7] comprise the only improvements to the approximation guarantee of GREEDY since the conjecture was first made.

Our results. We make progress on determining the optimal approximation guarantees of GREEDY and of another, more involved algorithm; the latter one improves the best proven approximation guarantee for SSP. In particular, we show the following theorems.

▶ **Theorem 1.** *The approximation guarantee of GREEDY is at most* $\frac{\sqrt{67}+2}{3} \approx 3.396$.

▶ **Theorem 2.** *An algorithm from the literature that combines GREEDY and a Max-ATSP approximation algorithm (outlined in Appendix A.2) computes a superstring of length at most* $\frac{\sqrt{67}+14}{9} \approx 2.466$ *times the optimal.*

Furthermore, our result implies improved approximation guarantees for two algorithms which are variants of GREEDY established in [3], namely TGREEDY and MGREEDY (outlined in Appendix A.2).

As in previous work, all our improved approximation bounds follow from a better inequality that relates certain overlaps between strings to the cost of the optimal solution.

2 The General Setting and Our Technical Contribution

Preliminaries. The set of input strings is denoted by $S = \{s_1, ..., s_{|S|}\}$. Without loss of generality, it is assumed that no string of S is a substring of another string of S. The *length* of a string s is the number of its characters and we denote it by $|s| \in \mathbb{Z}^+$. The concatenation of two strings s and t is denoted by st. A substring of s starting at character i and ending at character $j \geq i$ of s is denoted by $s[i, j]$.

By $\mathsf{ov}(s, t)$ we denote the maximum overlap to merge a string s to the left of a string $t \neq s$, i.e., the longest suffix of s that is a prefix of t. By $\mathsf{ov}(s, s)$ we denote the maximum self-overlap of string s with itself, which is smaller than $|s|$. By $\mathsf{pref}(s, t)$ we denote the prefix of s that remains after removing the overlap with t; thus, $s = \mathsf{pref}(s, t)\mathsf{ov}(s, t)$ and $|\mathsf{pref}(s, t)| = |s| - |\mathsf{ov}(s, t)|$.

2.1 Overlap Graph, Cycle-Closing Edges, and Overlap Inequalities

The overlap graph G_{ov} plays a central role in SSP approximation, including the analysis of GREEDY. It is a complete directed graph with self-loops in which vertices correspond to the input strings, and the weight of each edge (s,t) equals the overlap length $|\mathsf{ov}(s,t)|$.

Note that the optimal solution OPT for a fixed input corresponds to an optimal (maximum overlap) Hamiltonian path in G_{ov}; however, finding such a path is in general a hard problem. On the other hand, finding an optimal cycle cover CC in G_{ov} can be done efficiently. In particular, in a variant of GREEDY, called MGREEDY, such a cycle cover is produced as a by-product. Observe that the total overlap of edges in CC is only larger than that of the optimal Hamiltonian path OPT; indeed, by adding the edge between the endpoints of OPT, we obtain a Hamiltonian cycle, which is a particular cycle cover (not necessarily optimal).

The GREEDY algorithm can be stated as a heuristic for a Hamiltonian path in G_{ov}: Sort the edges of G_{ov} by their overlap lengths non-increasingly, then go over the sorted list and add the i-th edge e_i to the path unless:

- **(i)** there would be a vertex of indegree or outdegree more than one after adding e_i (that is, edge e_i shares a head node or a tail node with an edge picked in a previous step), or
- **(ii)** e_i closes a cycle.

The crucial difference between GREEDY for computing an approximate superstring and MGREEDY for the optimal cycle cover CC is the condition (ii), not present in the latter, i.e., MGREEDY is defined just by condition (i). Call an edge of CC *cycle-closing* if it is the last edge of its cycle added by MGREEDY to CC (i.e., it has the smallest overlap on the cycle, breaking ties arbitrarily).

To obtain a bound on the approximation guarantee of GREEDY, we intuitively need a suitable upper bound on the total overlap of cycle-closing edges, denoted o (strictly speaking, when analyzing GREEDY we consider only the optimal cycle cover of a certain subset of nodes in G_{ov}, but this does not make a difference for our technical contribution; we explain these details in Appendix A.1). Furthermore, the overlap bound should be in terms of the *length* (and not overlap) of OPT.

This intuition was formalized in [3], who proved that $o \leq 3 \cdot n$, where n is the length of the optimal solution OPT. Moreover, they show that such a bound is sufficient for a constant upper bound on the approximation ratio of GREEDY. Later works improved the inequality to $o \leq 2.5 \cdot n$ [11] and to $o < 2.425 \cdot n$ [7]. Our technical contribution is to show that $o < 2.396 \cdot n$.

In fact, these overlap inequalities are proven and applied in a stronger form of $o < n + \beta \cdot w$, where w is a lower bound on n. To define w, we associate each edge (s,t) of the overlap graph G_{ov} also with a *length* which equals the prefix length $|\mathsf{pref}(s,t)| = |s| - |\mathsf{ov}(s,t)|$. Then w is the total length of all edges in the optimal cycle cover CC.

2.2 Main technical result

We now state our main technical contribution.

▶ **Theorem 3.** *Let S be any input set of strings, and consider an optimal superstring of length n and an optimal cycle cover CC of length w, computed using MGREEDY. Let o be the sum of overlaps of all cycle-closing edges of CC. Then it holds that*

$$o \leq n + \beta \cdot w \qquad \text{for } \beta = (\sqrt{67} - 4)/3 \approx 1.396$$

The proofs of Theorems 1 and 2 using Theorem 3 are the same as in previous work, but we provide an outline for completeness. In Appendix A.1 we describe how Theorem 3 implies the improved upper bound on the approximation guarantees of GREEDY, using another inequality from Blum et al. [3]. Then, in Appendix A.2, we show how to derive better approximation guarantees for a family of SSP algorithms that are based on a Max-ATSP approximation algorithm; the argument is the same as in previous work (e.g., see [4, 15, 16, 7]).

2.3 Overview of the proof of Theorem 3

We build on our previous work [7], where one of the conceptual contributions was in classifying the cycles of CC into three main types. To define them, for a cycle c of CC we let

- $o(c)$ = the overlap of the cycle-closing edge of c, i.e., the smallest overlap on cycle c, and
- $w(c)$ = the total length of edges on c, i.e., the sum of prefixes of the edges of c.

The classification is done according to the $o(c)/w(c)$ ratio.

▶ **Definition 4.** *For parameter β defined in Theorem 3, a cycle c of CC is*

- extra large*, if $o(c) \leq \beta \cdot w(c)$,*
- large*, if $\beta \cdot w(c) < o(c) \leq 2w(c)$, and*
- small*, if $2w(c) < o(c)$.*

The intuition behind the names is that short cycles contain highly periodic strings (e.g., *abcabcabca*), whereas strings in large cycles are not so periodic (e.g., *abcdeabcd*)

In order to prove that $o \leq n + \beta \cdot w$ for $\beta = (\sqrt{67} - 4)/3$, we will assume, without loss of generality, that CC contains no extra large cycle. This follows by the argument in [7, Section 5.1], though for a different overlap to length ratio threshold between large and extra large cycles (which was suitably chosen to match the upper bound $o \leq n + 1.425w$). For completeness, we repeat the proof in Appendix B.

Our analysis in [7] proceeds by showing two incomparable bounds: one better if large cycles have much larger total length than small cycles, and another one for the other case. Namely, letting w_s be the sum of lengths of all small cycles and w_ℓ be the sum of lengths of large cycles, the first upper bound is

$$o \leq n + w_s + 1.5w_\ell \tag{1}$$

and the second upper bound is

$$o \leq n + w_\ell + \frac{31 + 3 \cdot \sqrt{57}}{14} w_s \approx n + w_\ell + 3.832w_s \ . \tag{2}$$

Using the better of (1) and (2) together with $w = w_s + w_\ell$, it follows that $o \leq n + 1.425w$ (recall that the extra large cycles are not taken into account here).

Our improvement and simplification comes from a better version of the second upper bound. Specifically, we show

$$o \leq n + w_\ell + (\gamma - 1) \cdot w_s \approx n + w_\ell + 2.884w_s \ , \tag{3}$$

where $\gamma = (\sqrt{67} + 19)/7 \approx 3.884$. In [7], the bound was shown by first modifying the input in such a way that the overlap graph G_{ov} has the property that all short cycles in the optimal cycle cover only consist of a single edge that is a self-loop. The analysis is then done utilizing this somewhat simpler cycle cover. However, the modification of the input introduces an additional loss that has to be accounted for in the bound. Our analysis is more direct and works with the original optimal cycle cover, which eliminates the need for the

input modification and therefore the additional loss. This brings new technical complications because certain properties no longer hold in these more general cycle covers. Nevertheless, we are able to provide a slightly simpler and more straightforward analysis.

Choice of parameters. To combine the two incomparable bounds, $o \leq n + w_s + 1.5 \cdot w_\ell$ and $o \leq n + (\gamma - 1) \cdot w_s + w_\ell$, we set $\lambda = \frac{1}{2\gamma - 3}$. As long as $\gamma \geq 2$, this means $\lambda \in [0, 1]$. We then multiply the first bound by $(1 - \lambda)$ and the second bound by λ and add them together. Using $w_s + w_\ell = w$ we get $o \leq n + (\frac{3}{2} - \frac{1}{4\gamma - 6}) \cdot w$. In Theorem 3, we want to show that $o \leq n + \beta \cdot w$ and so if

$$\frac{3}{2} - \frac{1}{4\gamma - 6} \leq \beta \tag{4}$$

we are done. We will also need

$$3 \cdot (\beta - \frac{2}{\gamma - 2}) \geq 1 \text{ (for Lemma 6)} \tag{5}$$

or equivalently

$$\gamma \geq 2 + \frac{6}{3\beta - 1} \text{ (for Lemma 12(b))}. \tag{6}$$

The maximum of these two lower bounds (4) and (5) on β is minimized for $\gamma = (\sqrt{67}+19)/7$ and at this point both bounds are equal to $(\sqrt{67} - 4)/3$, which is our choice for β. Apart from this, we will use a number of further inequalities that hold for this choice of parameters (but are not tight). Namely,

$$\frac{5}{2} + \frac{1}{2(\beta - 1)} \leq \gamma \text{ (for Lemma 12(c))}\ , \tag{7}$$

$$\beta \geq \frac{\gamma}{\gamma - 1} \text{ (for Lemma 12(d))}\ , \text{ and} \tag{8}$$

$$\gamma \geq 2 \text{ (for Lemma 12(d))}\ . \tag{9}$$

3 Analysis

In this section we show our improved second bound $o \leq n + w_\ell + (\gamma - 1) \cdot w_s$, following a similar general strategy as in [7].

3.1 Proof Outline

Consider a directed Hamiltonian cycle CC_0 of maximum total overlap in G_{ov}. This cycle is in particular also a (not necessarily maximum) cycle cover. Therefore, the total overlap of CC_0 must be bounded from above by the total overlap of CC. Our goal is to show something stronger than this: that there is a gap between the total overlap of CC_0 and the total overlap of CC that depends in a specific way on the properties of the cycles in CC. Specifically, let $\mathcal{L}$ and $\mathcal{S}$ denote the sets of large and small cycles in CC, respectively, and let $|\mathsf{CC}_i|$ denote the total overlap of a cycle cover CC_i. Then we want to show that the total overlap $|\mathsf{CC}|$ of CC is by at least

$$\sum_{c \in \mathcal{S}} \Big(o(c) - \gamma \cdot w(c) \Big) + \sum_{c \in \mathcal{L}} (o(c) - 2 \cdot w(c)) \tag{10}$$

larger than the total overlap $|\mathsf{CC}_0|$ of CC_0. Showing this is sufficient to establish $o \leq n + w_\ell + (\gamma - 1) \cdot w_s$ because

$$\begin{aligned} n &\geq \sum_{\ell=1}^{|S|} |s_\ell| - |\mathsf{CC}_0| \geq \sum_{\ell=1}^{|S|} |s_\ell| - |\mathsf{CC}| + \sum_{c \in \mathcal{S}} (o(c) - \gamma \cdot w(c)) + \sum_{c \in \mathcal{L}} (o(c) - 2 \cdot w(c)) \\ &\geq \sum_{c \in \mathcal{S}} w(c) + \sum_{c \in \mathcal{L}} w(c) + \sum_{c \in \mathcal{S}} (o(c) - \gamma \cdot w(c)) + \sum_{c \in \mathcal{L}} (o(c) - 2 \cdot w(c)) \\ &= \sum_{c \in \mathcal{S}} (o(c) - (\gamma - 1) \cdot w(c)) + \sum_{c \in \mathcal{L}} (o(c) - w(c)) \\ &= o - (\gamma - 1) \cdot \sum_{c \in \mathcal{S}} w(c) - \sum_{c \in \mathcal{L}} w(c) = o - (\gamma - 1) \cdot w_s - w_\ell \,. \end{aligned}$$

Related cycles. Before proceeding to describe how we show (10), we borrow the following definition of *related cycles* from [7] that is useful to improve our final bounds slightly. We note that a simpler version of our proof could still be carried out without this additional concept, but at the cost of a slightly weaker bound.

▶ **Definition 5.** *We define a relation R between cycles as follows. A* small *cycle c of CC is related to a* large *cycle c' of CC if $w(c) \leq (\beta/2 - 1/6) \cdot w(c')$ and there exists an edge e in G_{ov} that has one endpoint in cycle c, the other endpoint in cycle c' and satisfies $|\mathsf{ov}(e)| \geq \beta \cdot w(c')$. In this case, we write $(c, c') \in R$.*

In [7], the following lemma is shown. We use different values for β and γ, but the proof of the lemma only requires that $3 \cdot (\beta - 2/(\gamma - 2)) \geq 1$ and this is still satisfied for our new choice of $\beta = (\sqrt{67} - 4)/3$ and $\gamma = (5 - 3\beta)/(3 - 2\beta)$.

▶ **Lemma 6** (Lemma 7.3 in [7])**.** *For every large cycle c' of CC, at most two different small cycles of CC are related to c'.*

Transforming cycle cover CC_0 into CC in small steps. We analyze the difference of the total overlap between CC_0 and CC in small steps, gradually changing the Hamiltonian cycle CC_0 into a sequence of cycle covers $\mathsf{CC}_0, \mathsf{CC}_1, \mathsf{CC}_2, \ldots$ until we obtain CC. We modify a cycle cover CC_i by removing two edges $f = (v', v)$ and $f' = (u, u')$ from $\mathsf{CC}_i \setminus \mathsf{CC}$ and replace them with the new edges $e = (u, v)$ and $e' = (v', u')$. The resulting set of edges forms a (not necessarily optimal) cycle cover again. Furthermore, if the edges are chosen such that $e \in \mathsf{CC}$ or $e' \in \mathsf{CC}$ (or both), then the resulting cycle cover is closer to the cycle cover CC in the sense that the cardinality of the symmetric difference of the corresponding edge sets decreases.

For a cycle cover CC_i, let $\mathcal{M}(\mathsf{CC}_i)$ be the set of *small* cycles c in CC for which CC_i contains no edge with one endpoint in c and the other endpoint being a string not in c. We define

$$\begin{aligned} \phi(i) = \sum_{c \in \mathcal{M}(\mathsf{CC}_i)} \Bigg(&\min\{|\mathsf{ov}(\hat{e})| \mid \hat{e} \in \mathsf{CC}_i \text{ connects two strings of } c\} - \gamma \cdot w(c) \\ &- \sum_{c' : (c, c') \in R} \left(w(c') - \frac{o(c')}{2} \right) \Bigg) \,. \end{aligned}$$

The idea is to perform such edge swaps to obtain a sequence $\mathsf{CC}_0, \mathsf{CC}_1, \mathsf{CC}_2, \ldots, \mathsf{CC}_k = \mathsf{CC}$ of cycle covers, such that each cycle cover CC_i is closer to CC than the previous one CC_{i-1} and such that $|\mathsf{CC}_i| \geq |\mathsf{CC}_0| + \phi(i)$. Then this implies (10) since

$$
\begin{aligned}
|\mathsf{CC}| - |\mathsf{CC}_0| = |\mathsf{CC}_k| - |\mathsf{CC}_0| &\geq \phi(k) \\
&= \sum_{c \in \mathcal{M}(\mathsf{CC})} \Big(\min\{|\mathsf{ov}(\hat{e})| \mid \hat{e} \in \mathsf{CC} \text{ connects two strings of } c\} - \gamma \cdot w(c) \\
&\quad - \sum_{c' : (c,c') \in R} \Big(w(c') - \frac{o(c')}{2} \Big) \Big) \\
&= \sum_{c \in \mathcal{S}} \Big(\min\{|\mathsf{ov}(\hat{e})| \mid \hat{e} \in \mathsf{CC} \text{ connects two strings of } c\} - \gamma \cdot w(c) \\
&\quad - \sum_{c' : (c,c') \in R} \Big(w(c') - \frac{o(c')}{2} \Big) \Big) \\
&= \sum_{c \in \mathcal{S}} \Big(o(c) - \gamma \cdot w(c) - \sum_{c' : (c,c') \in R} \Big(w(c') - \frac{o(c')}{2} \Big) \Big) \\
&= \sum_{c \in \mathcal{S}} (o(c) - \gamma \cdot w(c)) - \sum_{c \in \mathcal{S}} \sum_{c' : (c,c') \in R} \Big(w(c') - \frac{o(c')}{2} \Big) \\
&\geq \sum_{c \in \mathcal{S}} (o(c) - \gamma \cdot w(c)) - \sum_{c \in \mathcal{L}} (2 \cdot w(c) - o(c)) \,,
\end{aligned}
$$

where the last step follows from Lemma 6 and the fact that for large cycles c', by definition, $2w(c') \geq o(c')$.

We use induction to show that it is possible to construct the desired sequence of cycle covers that satisfies $|\mathsf{CC}_i| \geq |\mathsf{CC}_0| + \phi(i)$. The base case is $i = 0$ and we have $\phi(i) = 0$ because $\mathcal{M}(\mathsf{CC}_0) = \emptyset$. (Strictly speaking, it may happen that $\mathcal{M}(\mathsf{CC}_0) \neq \emptyset$; however, in such a case, the optimal Hamiltonian cycle CC_0 is a small cycle of CC, thus $\mathsf{CC}_0 = \mathsf{CC}$. Moreover, in such a case, (1) implies $o < n + w$.)

In the following, we assume that we have a cycle cover CC_i with $|\mathsf{CC}_i| \geq |\mathsf{CC}_0| + \phi(i)$ and we show how to construct CC_{i+1} such that $|\mathsf{CC}_{i+1}| \geq |\mathsf{CC}_0| + \phi(i+1)$ and such that the symmetric difference between CC_{i+1} and CC is smaller than the symmetric difference between CC_i and CC. Specifically, we will identify a swap of four edges as described above to obtain CC_{i+1} from CC_i such that:

- one of the edges that are swapped in belongs to CC, which implies that the symmetric difference between CC_{i+1} and CC will decrease, and
- $|\mathsf{CC}_{i+1}| - |\mathsf{CC}_i| \geq \phi(i+1) - \phi(i)$.

This proves the claim due to the induction hypothesis.

3.2 Important Lemmas

We begin with the following bound on the overlap between two strings from different cycles of CC.

▶ **Lemma 7** (Lemma 9 in [3]). *Let c and $c' \neq c$ be two cycles in CC. It holds that $|\mathsf{ov}(s, s')| < w(c) + w(c')$ for any two strings $s \in c$ and $s' \in c'$.*

When changing cycle cover CC_i into CC_{i+1}, we identify an edge $e = (u, v) \in \mathsf{CC} \setminus \mathsf{CC}_i$ that we add into CC_{i+1}. This triggers removal of edges $f = (v', v)$ and $f' = (u, u')$ from CC_i and addition of one more edge $e' = (v', u')$ that does not belong to CC_i but may or may not be in CC; see Figure 1. In the following, we provide several lower bounds on $|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')|$, which is the total overlap length difference between CC_i and CC_{i+1}. The first lemma is the well-known *Monge Condition*.

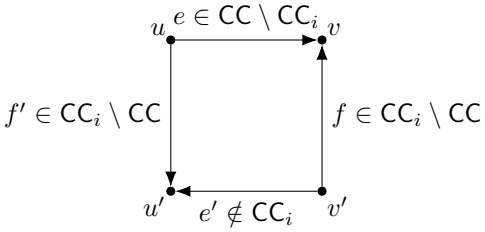


Figure 1 Illustration of the notation used in lemmas in Section 3.2.

▶ **Lemma 8** (Lemma 7 in [3]). *Let $e = (u, v)$, $f = (v', v)$, $f' = (u, u')$, $e' = (v', u')$ be edges in G_{ov}, such that $\max\{|\mathsf{ov}(e)|, \mathsf{ov}(e')|\} \geq \max\{|\mathsf{ov}(f)|, |\mathsf{ov}(f')|\}$. Then $|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| \geq 0$.*

The following lemma is shown in [7, Lemma 7.5] for the special case of inputs where each small cycle of CC consists of one string. Below, we generalize it for any input and cycle.

▶ **Lemma 9.** *Let $e = (u, v)$, $f = (v', v)$, $f' = (u, u')$, and $e' = (v', u')$ be edges in G_{ov} such that e is an edge in cycle c in CC. Then,*

$$|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| > |\mathsf{ov}(e)| - \max\{|\mathsf{ov}(f)|, |\mathsf{ov}(f')|\} - w(c) \ .$$

Before proving Lemma 9, we recall a few definitions from the literature. Consider a cycle c of CC having k nodes $s_1, s_2, \ldots, s_k$. Assuming that the cycle-closing edge of c is (s_k, s_1), we define $s(c)$ as the string $\mathsf{pref}(s_1, s_2)\mathsf{pref}(s_2, s_3) \ldots \mathsf{pref}(s_k, s_1)$.

A *semi-infinite* string is a string obtained by concatenating an infinite number of finite strings. A semi-infinite string s is *periodic* if $s = ts$ for a non-empty string t, that is, $s = t^\infty$.

A string t is a *factor* of a string s if $s = t^i y$ for an integer $i > 0$, where y is a (possibly empty) prefix y of t. By $\mathsf{factor}(s)$ of s, we denote the shortest factor of s and we define $\mathsf{period}(s) = |\mathsf{factor}(s)|$. Finally, we say that a string s has a *periodicity* of length q for $q \leq |s|$ if s is a prefix of the semi-infinite string x^∞ for some string x of length q.

Next, we need a basic observation.

▶ **Observation 10.** *Let s and t be two strings that are substrings of some string z. Then, $|\mathsf{ov}(s, t)| > \min\{|s|, |t|\} - \mathsf{period}(z)$.*

Proof. We can assume without loss of generality (w.l.o.g.) that $|s| \leq |t|$. This is because, otherwise, let s_R, t_R, and z_R be the reverse of the strings s, t, and z, respectively. We observe that $\mathsf{ov}(t_R, s_R) = \mathsf{ov}(s, t)$ and $\mathsf{period}(z_R) = \mathsf{period}(z)$. Clearly also $|s_R| = |s|$, $|t_R| = |t|$. Therefore, the inequality in the statement of the observation is equivalent to $|\mathsf{ov}(t_R, s_R)| > \min\{|s_R|, |t_R|\} - \mathsf{period}(z_R)$. Hence, if $|s| > |t|$ then $|t_R| \leq |s_R|$ and we can apply the arguments below to the strings t_R, s_R, and z_R instead of s, t, and z (in this order).

Since s and t are substrings of z we can write them as $s = z[i, i + |s| - 1]$ and $t = z[j, j+|t|-1]$ for some i and j. Because of the period of z, we can assume that $i \in [1, \mathsf{period}(z)]$ and $j \in [1, \mathsf{period}(z)]$.

- If $j \geq i$, we have $\mathsf{ov}(s, t) = z[j, i+|s|-1]$ and hence $|\mathsf{ov}(s, t)| = i - j + |s| > |s| - \mathsf{period}(z)$.
- If $j < i$ and $j + \mathsf{period}(z) > |z|$, then $j + \mathsf{period}(z) > |z| \geq i + |s| - 1$ and hence, $|\mathsf{ov}(s, t)| \geq 0 > j - i \geq |s| - \mathsf{period}(z)$.
- If $j < i$ and $j + \mathsf{period}(z) \leq |z|$, we observe that $t = z[j, j + |t| - 1]$ also has $z[j + \mathsf{period}(z), \min\{j + |t| - 1 + \mathsf{period}(z), |z|\}]$ as a prefix (indeed, if $j + |t| - 1 + \mathsf{period}(z) \leq |z|$ this is not just a prefix of t, but exactly t). Since $i \leq j + \mathsf{period}(z)$ and $|s| \leq |t|$, we have $\mathsf{ov}(s, t) = z[j + \mathsf{period}(z), i + |s| - 1]$ and hence, $|\mathsf{ov}(s, t)| = i - j + |s| - \mathsf{period}(z) > |s| - \mathsf{period}(z)$. ◀

Proof of Lemma 9. Since $\mathsf{ov}(f)$ and $\mathsf{ov}(f')$ are substrings of $s(c)^\infty$, we use Observation 10 to get

$$\begin{aligned}|\mathsf{ov}(e')| &\geq |\mathsf{ov}(\mathsf{ov}(f),\mathsf{ov}(f'))|\\ &> \min\{|\mathsf{ov}(f)|,|\mathsf{ov}(f')|\} - \mathsf{period}(s(c)^\infty) \geq \min\{|\mathsf{ov}(f)|,|\mathsf{ov}(f')|\} - w(c)\,.\end{aligned}$$

It follows that

$$\begin{aligned}&|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')|\\ &\quad> |\mathsf{ov}(e)| + \min\{|\mathsf{ov}(f)|,|\mathsf{ov}(f')|\} - w(c) - |\mathsf{ov}(f)| - |\mathsf{ov}(f')|\\ &\quad= |\mathsf{ov}(e)| - \max\{|\mathsf{ov}(f)|,|\mathsf{ov}(f')|\} - w(c)\,.\end{aligned}$$

◀

The following lemma is, also, due to [7]. Here, we state it in a slightly different way, but the proof is essentially the same and included in Appendix C for completeness.

▶ **Lemma 11.** *Consider the edges $e = (u, v)$, $f = (v', v)$, $f' = (u, u')$, and $e' = (v', u')$ between (not necessarily different) nodes u, u', v, v' in G_{ov}. Suppose u' and v' are strings in the same cycle c' of CC and that whichever of f or f' has larger overlap connects a string from cycle c and a string from cycle $c' \neq c$ (if $|\mathsf{ov}(f)| = |\mathsf{ov}(f')|$ then it is sufficient if one of them satisfies this). If $|\mathsf{ov}(e)| \geq w(c) + w(c')$, then*

$$|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| > |\mathsf{ov}(e')| - w(c')\,.$$

The following lemma draws conclusions from the previous ones in a way that will be useful later for our analysis.

▶ **Lemma 12.** *Consider the edges $e = (u, v)$, $f = (v', v)$, $f' = (u, u')$, and $e' = (v', u')$ between (not necessarily different) nodes u, u', v, v' in G_{ov}. Suppose e is an edge in a cycle c of CC. Suppose further that $|\mathsf{ov}(e)| \geq \max\{|\mathsf{ov}(f)|,|\mathsf{ov}(f')|\}$ and the edge of f and f' that has larger overlap connects a string of cycle c and a string of cycle $c' \neq c$ (if $|\mathsf{ov}(f)| = |\mathsf{ov}(f')|$, then either one of f and f' may satisfy this condition). All of the following statements hold:*

(a) $|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| \geq 0$.

(b) *If $w(c) \geq (\beta/2 - 1/6) \cdot w(c')$, then $|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| \geq |\mathsf{ov}(e)| - \gamma w(c)$.*

(c) *If $w(c) \geq (\beta - 1) \cdot w(c')$, then $|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| \geq |\mathsf{ov}(e)| - \gamma w(c) - w(c')/2 + w(c)/2$.*

(d) *Furthermore, if v' and u' are strings in the same cycle in CC, then also $|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| \geq \max\{|\mathsf{ov}(e')| - \gamma w(c'), |\mathsf{ov}(e)| - \gamma w(c) + |\mathsf{ov}(e')| - \gamma w(c')\}$.*

Proof. We show the relevant lower bounds on $|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')|$ separately.

(a) Due to Lemma 8, we have $|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| \geq 0$.

(b) If $w(c) \geq (\beta/2 - 1/6) \cdot w(c')$, due to Lemma 9, we have

$$\begin{aligned}|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| &\geq |\mathsf{ov}(e)| - \max\{|\mathsf{ov}(f)|,|\mathsf{ov}(f')|\} - w(c)\\ &\geq |\mathsf{ov}(e)| - 2w(c) - w(c') \geq |\mathsf{ov}(e)| - \gamma w(c)\,,\end{aligned}$$

where the second step uses Lemma 7 and the last inequality follows from $2+6/(3\beta-1) = \gamma$.

(c) If $w(c) \geq (\beta - 1) \cdot w(c')$, we have due to Lemma 9 that

$$\begin{aligned}|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| &\geq |\mathsf{ov}(e)| - \max\{|\mathsf{ov}(f)|,|\mathsf{ov}(f')|\} - w(c)\\ &\geq |\mathsf{ov}(e)| - 2w(c) - w(c')\\ &= |\mathsf{ov}(e)| - \frac{5}{2}w(c) - w(c')/2 - w(c')/2 + w(c)/2\\ &\geq |\mathsf{ov}(e)| - \gamma w(c) - w(c')/2 + w(c)/2\,,\end{aligned}$$

where the second step uses Lemma 7 and the last inequality follows from $5/2 + 1/(2(\beta - 1)) \leq \gamma$.

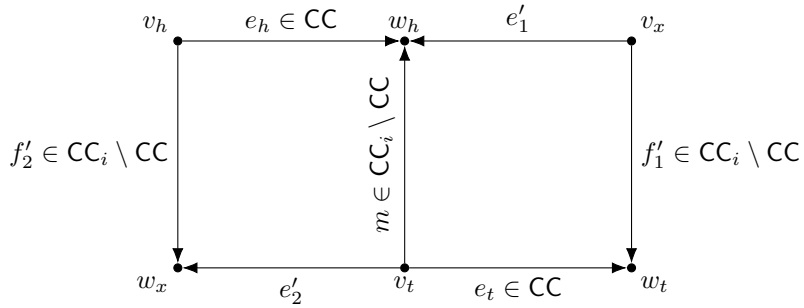


Figure 2 Illustration of the notation. Note that we also allow nodes to be equal to one another here, e.g., it could be that $w_t = w_x$, in which case $e_t = e'_2$, $v_h = v_x$, $e_h = e'_1$, and $f'_1 = f'_2$.

(d) - Suppose v' and u' are strings in the same cycle in CC. If $|\mathsf{ov}(e)| \geq w(c)+w(c')$, we apply Lemma 11 to get $|\mathsf{ov}(e)|+|\mathsf{ov}(e')|-|\mathsf{ov}(f)|-|\mathsf{ov}(f')| \geq |\mathsf{ov}(e')|-w(c') \geq |\mathsf{ov}(e')|-\gamma w(c')$. Otherwise, we have $|\mathsf{ov}(e)| < w(c) + w(c')$ and hence,

$$\begin{aligned} |\mathsf{ov}(f)| \leq w(c) + w(c') &= w(c') + \gamma w(c) - (\gamma-1)w(c) \\ &\leq w(c') + (\gamma-1)o(c) - (\gamma-1)w(c) \\ &\leq w(c') + (\gamma-1)|\mathsf{ov}(e)| - (\gamma-1)w(c) < \gamma w(c')\,, \end{aligned}$$

since it holds $\beta \geq \frac{\gamma}{\gamma-1}$ and $o(c) > \beta w(c)$ for any large or small cycle c (recall that we assume that CC contains no extra large cycle). We get $|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| \geq |\mathsf{ov}(e')| - |\mathsf{ov}(f)| \geq |\mathsf{ov}(e')| - \gamma w(c')$.

- Suppose v' and u' are strings in the same cycle in CC. Due to Lemma 7,

$$\begin{aligned} |\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(f)| - |\mathsf{ov}(f')| &\geq |\mathsf{ov}(e)| + |\mathsf{ov}(e')| - 2\max\{|\mathsf{ov}(f)|, |\mathsf{ov}(f')|\} \\ &\geq |\mathsf{ov}(e)| - 2w(c) + |\mathsf{ov}(e')| - 2w(c') \\ &\geq |\mathsf{ov}(e)| - \gamma w(c) + |\mathsf{ov}(e')| - \gamma w(c')\,. \end{aligned}$$

◀

3.3 The Induction Step

We specify how an edge swap is made at a fixed step i in which we obtain cycle cover CC_{i+1} from CC_i. We start by identifying the largest-overlap edge $m = (v_t, w_h)$ in $\mathsf{CC}_i \setminus \mathsf{CC}$, breaking ties arbitrarily. Six further edges will be important. First, let $e_h = (v_h, w_h)$ and $e_t = (v_t, w_t)$ be the edges in CC that share heads and tails with m, respectively. Further, let $f'_1 = (v_x, w_t)$ and $f'_2 = (v_h, w_x)$ be the two edges in $\mathsf{CC}_i \setminus \mathsf{CC}$ that share heads with e_t and tails with e_h, respectively. Lastly, define $e'_1 = (v_x, w_h)$ and $e'_2 = (v_t, w_x)$. See Figure 2 for a summary of this notation. It is important to note that the six strings v_h, w_h, v_x, w_x, v_t, and w_t are not necessarily different.

With this, we can define two potential edge swaps. In the first one, we add e_t and e'_1 to the cycle cover and instead remove m and f'_1. In the second one, we add e_h and e'_2 to the cycle over and instead remove m and f'_2. Which one of these two swaps we will perform depends on a few properties of the edges involved. First of all, we assume that $|\mathsf{ov}(e_h)| \geq |\mathsf{ov}(e_t)|$. Otherwise, all the remaining arguments follow symmetrically by considering e_t instead of e_h and vice versa. Furthermore, we have that

$$|\mathsf{ov}(e_h)| \geq |\mathsf{ov}(m)|\,, \tag{11}$$

since otherwise $|\mathsf{ov}(m)| > |\mathsf{ov}(e_h)| \geq |\mathsf{ov}(e_t)|$ and m would be added to CC by the greedy algorithm for the optimal cycle cover before e_h and e_t, contradicting the choice of m as an edge of largest overlap in $\mathsf{CC}_i \setminus \mathsf{CC}$.

We observe that there are two reasons why $\phi(i+1)$ may be larger than $\phi(i)$.

- The first potential reason is a difference between the sets $\mathcal{M}(\mathsf{CC}_{i+1})$ and $\mathcal{M}(\mathsf{CC}_i)$. We know that $\mathcal{M}(\mathsf{CC}_{i+1}) \supseteq \mathcal{M}(\mathsf{CC}_i)$, because if a cycle c is in $\mathcal{M}(\mathsf{CC}_i)$, then there is no edge in CC_i connecting a string of c to a string of another cycle. That means that the edges f and f' that we remove from CC_i in the process of constructing CC_{i+1} either have both their endpoints in c or both their endpoints not in c. If both endpoints of both edges f and f' are part of c, then also the two edges that are swapped in to obtain CC_{i+1} from CC_i have their endpoints entirely in c. Therefore, c would still be in $\mathcal{M}(\mathsf{CC}_{i+1})$ after the swap. If both endpoints of both edges f and f' are outside of c, then also the two edges that are swapped in to obtain CC_{i+1} from CC_i have their endpoints entirely outside of c. Again, c would still be in $\mathcal{M}(\mathsf{CC}_{i+1})$ after the swap in this case. Finally, if one of f and f' has both endpoints in c and the other one has both endpoints outside of c, then the two edges that are swapped in both have one endpoint in c and the other endpoint outside of c. However, this is not possible because one of the edges we swap in is e_h or e_t and must therefore be part of the optimal cycle cover CC.

  We can further observe that $\mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i)$ must either be equal to $\emptyset$, $\{c\}$, $\{c'\}$, or $\{c, c'\}$, where c and c' are the cycles that e_h and e_t belong to in CC, respectively. (It is possible that $c = c'$.) To see this, observe that one edge being swapped out to obtain CC_{i+1} from CC_i is m and that m has one endpoint (w_h) in c and the other endpoint (v_t) in c'. However, for each cycle of CC, it is clear from a parity argument that the number of edges of CC_i connecting the cycle to other cycles must be even. Hence, for a cycle c'' to be in $\mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i)$, each of the edges being swapped out must have a string from cycle c'' as an endpoint. This can only be true for c or c' and not for any other cycle.

  Overall, if this reason for the difference between $\phi(i+1)$ and $\phi(i)$ applies, we have that

$$\phi(i+1) - \phi(i) = \sum_{c \in \mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i)} \Big(\min\{|\mathsf{ov}(\hat{e})| \mid \hat{e} \in \mathsf{CC}_i \text{ connects two strings of } c\} - \gamma \cdot w(c) - \sum_{c' : (c,c') \in R} \Big(w(c') - \frac{o(c')}{2} \Big) \Big) .$$

- The second potential reason why $\phi(i+1)$ may be larger than $\phi(i)$ is that for a cycle $c \in \mathcal{M}(\mathsf{CC}_i)$ the term $\min\{|\mathsf{ov}(\hat{e})| \mid \hat{e} \in \mathsf{CC}_i \text{ connects two strings of } c\}$ could change. However, this can only happen if $\mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i) = \emptyset$ and, furthermore, it can only happen for a cycle c when both edges f and f' that are swapped out have both their endpoints in cycle c. In this case, all four strings involved in the swap (either v_h, w_x, w_h, and v_t or v_x, w_t, w_h, and v_t), must be part of the same cycle in CC. If the value $\min\{|\mathsf{ov}(\hat{e})| \mid \hat{e} \in \mathsf{CC}_{i+1} \text{ connects two strings of } c\}$ is larger than the value $\min\{|\mathsf{ov}(\hat{e})| \mid \hat{e} \in \mathsf{CC}_i \text{ connects two strings of } c\}$, then an edge in $\arg\min\{|\mathsf{ov}(\hat{e})| \mid \hat{e} \in \mathsf{CC}_i \text{ connects two strings of } c\}$ must have been swapped out. This means, that if f and f' are the edges being swapped out to obtain CC_{i+1} from CC_i, then $\min\{|\mathsf{ov}(\hat{e})| \mid \hat{e} \in \mathsf{CC}_i \text{ connects two strings of } c\} = \min\{|\mathsf{ov}(f)|, |\mathsf{ov}(f')|\}$. If e and e' are the two edges being swapped in, the new value of $\min\{|\mathsf{ov}(\hat{e})| \mid \hat{e} \in \mathsf{CC}_{i+1} \text{ connects two strings of } c\}$ can be at most $\min\{|\mathsf{ov}(e)|, |\mathsf{ov}(e')|\}$ because e and e' are in CC_{i+1} and satisfy the condition that they connect two strings of c. So overall, in this situation,

$$\phi(i+1) - \phi(i) \leq \min\{|\mathsf{ov}(e)|, |\mathsf{ov}(e')|\} - \min\{|\mathsf{ov}(f)|, |\mathsf{ov}(f')|\} .$$

In summary, we note that only one of the two reasons can apply for any fixed step i. If there is an increase of $\phi(i+1)$ over $\phi(i)$ due to the first reason (a change in the set $\mathcal{M}(\mathsf{CC}_{i+1})$ compared to $\mathcal{M}(\mathsf{CC}_i)$), then there is no increase due to the second reason and vice versa.

We are now ready to complete the proof by showing how to select one of the two identified swap operations such that the total overlap increases by at least $\phi(i+1)-\phi(i)$.

- If m connects two strings of the same cycle in CC, then observe that $\mathcal{M}(\mathsf{CC}_{i+1}) = \mathcal{M}(\mathsf{CC}_i)$. We swap in e_h and e'_2 and swap out f'_2 and m. Since $|\mathsf{ov}(e_h)| \geq |\mathsf{ov}(m)|$ by (11), we can apply Lemma 8 and establish that the total overlap does not decrease when this swap is performed.

  Furthermore, if v_h, w_h, v_t, and w_x all belong to the same cycle of CC, then the total overlap increases by $|\mathsf{ov}(e_h)| + |\mathsf{ov}(e'_2)| - |\mathsf{ov}(f'_2)| - |\mathsf{ov}(m)| \geq |\mathsf{ov}(e'_2)| - |\mathsf{ov}(f'_2)| \geq \min\{|\mathsf{ov}(e_h)|, |\mathsf{ov}(e'_2)|\} - \min\{|\mathsf{ov}(f'_2)|, |\mathsf{ov}(m)|\}$, where the second inequality uses $|\mathsf{ov}(f'_2)| \leq |\mathsf{ov}(m)|$ by the definition of m. This is the only case in which

  $$\min\{|\mathsf{ov}(e)| \mid e \text{ is edge of } \mathsf{CC}_i \text{ connecting two strings of cycle } c\}$$

  can change for a cycle in $c \in \mathcal{M}(\mathsf{CC}_i)$ and the increase is at least $\min\{|\mathsf{ov}(e_h)|, |\mathsf{ov}(e'_2)|\} - \min\{|\mathsf{ov}(f'_2)|, |\mathsf{ov}(m)|\} \geq \phi(i+1)-\phi(i)$, as required.
- If m connects strings of two different cycles in CC and $|\mathsf{ov}(e_t)| \geq |\mathsf{ov}(m)|$. Let c be the cycle of e_h and c' be the cycle of e_t. If $w(c) \geq w(c')$, we swap in $e = e_h$ and $e' = e'_2$ and swap out $f' = f'_2$ and m. Otherwise, we swap in $e = e_t$ and $e' = e'_1$ and swap out $f' = f'_1$ and m.

  We distinguish between these two cases:
  - Suppose $w(c) \geq w(c')$.

    Then, if $c' \in \mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i)$, Lemma 12(d) applies and we know that the increase in total overlap due to the swap is $|\mathsf{ov}(e)|+|\mathsf{ov}(e')|-|\mathsf{ov}(m)|-|\mathsf{ov}(f')| \geq \max\{|\mathsf{ov}(e')|-\gamma w(c'), |\mathsf{ov}(e)|-\gamma w(c)+|\mathsf{ov}(e')|-\gamma w(c')\} \geq \phi(i+1)-\phi(i)$, as required since $\phi(i+1)-\phi(i)$ is either equal to $|\mathsf{ov}(e')|-\gamma w(c')$ or equal to $|\mathsf{ov}(e)|-\gamma w(c)+|\mathsf{ov}(e')|-\gamma w(c')$ depending on whether $\mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i) = \{c'\}$ or $\mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i) = \{c', c\}$.

    Otherwise, if $c' \notin \mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i)$, Lemma 12(a) and (b) both apply and we know that the increase in total overlap due to the swap is $|\mathsf{ov}(e)|+|\mathsf{ov}(e')|-|\mathsf{ov}(m)|-|\mathsf{ov}(f')| \geq \max\{0, |\mathsf{ov}(e)| - \gamma w(c)\} \geq \phi(i+1)-\phi(i)$, as required since $\phi(i+1)-\phi(i)$ is either equal to 0 or equal to $|\mathsf{ov}(e)| - \gamma w(c)$ depending on whether $\mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i) = \emptyset$ or $\mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i) = \{c\}$.
  - Suppose $w(c) < w(c')$.

    Then, the same argument as above holds with the only difference being that the roles of e and e' and of c and c' are reversed. Specifically, if $c \in \mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i)$, Lemma 12(d) applies with the roles of e and e' and the roles of c and c' reversed. It follows that the increase in total overlap due to the swap is $|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(m)| - |\mathsf{ov}(f')| \geq \max\{|\mathsf{ov}(e)| - \gamma w(c), |\mathsf{ov}(e')| - \gamma w(c') + |\mathsf{ov}(e)| - \gamma w(c)\} \geq \phi(i+1) - \phi(i)$, as required.

    Otherwise, if $c \notin \mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i)$, Lemma 12(a) and (b) both apply (again with the roles of e and e' and c and c' reversed) and we know that the increase in total overlap due to the swap is $|\mathsf{ov}(e)| + |\mathsf{ov}(e')| - |\mathsf{ov}(m)| - |\mathsf{ov}(f')| \geq \max\{0, |\mathsf{ov}(e')| - \gamma w(c')\} \geq \phi(i+1) - \phi(i)$, as required.
- If m connects strings of two different cycles in CC and $|\mathsf{ov}(e_t)| < |\mathsf{ov}(m)|$, then we swap in e_h and e'_2 and swap out f'_2 and m. Let c be the cycle of e_h and c' be the cycle of e_t.
  - If $\mathcal{M}(\mathsf{CC}_{i+1}) = \mathcal{M}(\mathsf{CC}_i)$, then Lemma 12(a) shows that the total overlap does not decrease, while the potential $\phi(i)$ does not increase.

- If $c' \in \mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i)$, then w_x and v_t must both be strings in cycle c' as otherwise, v' is a string of cycle c' and w_x is a string of a different cycle and thus e'_2, which is an edge in CC_{i+1}, would connect a string of cycle c' to a string of another cycle. Thus, by Lemma 12(d), $|\mathsf{ov}(e_h)| + |\mathsf{ov}(e'_2)| - |\mathsf{ov}(m)| - |\mathsf{ov}(f'_2)| \geq \max\{|\mathsf{ov}(e'_2)| - \gamma w(c'), |\mathsf{ov}(e_h)| - \gamma w(c) + |\mathsf{ov}(e'_2)| - \gamma w(c')\} \geq \phi(i+1) - \phi(i)$, as required.
- If $\mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i) = \{c\}$ and $(c, c') \in R$, we first observe

$$w(c) \geq |\mathsf{ov}(m)| - w(c') > |\mathsf{ov}(e_t)| - w(c') \geq o(c') - w(c') \geq (\beta - 1) \cdot w(c') \,,$$

  where the third inequality follows from the fact that e_t is an edge of the cycle c' and the last step follows because c' is not extra large. Therefore, we can apply Lemma 12(c) which is sufficient because $w(c)/2 - w(c')/2 = w(c)/2 + w(c')/2 - w(c') \geq |\mathsf{ov}(m)|/2 - w(c') \geq o(c')/2 - w(c')$ and therefore, $|\mathsf{ov}(e_h)| + |\mathsf{ov}(e'_2)| - |\mathsf{ov}(m)| - |\mathsf{ov}(f'_2)| \geq |\mathsf{ov}(e)| - \gamma w(c) - w(c')/2 + w(c)/2 \geq |\mathsf{ov}(e)| - \gamma w(c) - w(c') + o(c')/2 \geq \phi(i+1) - \phi(i)$, as required.
- If $\mathcal{M}(\mathsf{CC}_{i+1}) \setminus \mathcal{M}(\mathsf{CC}_i) = \{c\}$ and $(c, c') \notin R$, there are two possibilities.
  1. If c' is a small cycle, then $w(c') \leq o(c') - w(c') \leq |\mathsf{ov}(e_t)| - w(c') < |\mathsf{ov}(m)| - w(c') \leq w(c)$, where the first step uses the definition of a small cycle and the last step uses Lemma 7.
  2. If c' is a large cycle and $(c, c') \notin R$, then, because $|\mathsf{ov}(m)| > |\mathsf{ov}(e_t)| \geq \beta w(c')$ by the definition of related cycles, $w(c) > (\beta/2 - 1/6) \cdot w(c')$.

  Either way $w(c) > (\beta/2 - 1/6) \cdot w(c')$, which means that Lemma 12(b) implies $|\mathsf{ov}(e_h)| + |\mathsf{ov}(e'_2)| - |\mathsf{ov}(m)| - |\mathsf{ov}(f'_2)| \geq |\mathsf{ov}(e_h)| - \gamma w(c) \geq \phi(i+1) - \phi(i)$, as required.

References

1. Chris Armen and Clifford Stein. Improved length bounds for the shortest superstring problem. In *Proceedings of the 4th International Workshop on Algorithms and Data Structures (WADS)*, pages 494–505, 1995. doi:10.1007/3-540-60220-8_88.
2. Chris Armen and Clifford Stein. A 2 2/3 superstring approximation algorithm. *Discret. Appl. Math.*, 88(1-3):29–57, 1998. doi:10.1016/S0166-218X(98)00065-1.
3. Avrim Blum, Tao Jiang, Ming Li, John Tromp, and Mihalis Yannakakis. Linear approximation of shortest superstrings. *Journal of the ACM*, 41(4):630–647, 1994. doi:10.1145/179812.179818.
4. Dany Breslauer, Tao Jiang, and Zhigen Jiang. Rotations of periodic strings and short superstrings. *J. Algorithms*, 24(2):340–353, 1997. doi:10.1006/jagm.1997.0861.
5. M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.
6. Artur Czumaj, Leszek Gasieniec, Marek Piotrów, and Wojciech Rytter. Sequential and parallel approximation of shortest superstrings. *J. Algorithms*, 23(1):74–100, 1997. doi:10.1006/jagm.1996.0823.
7. Matthias Englert, Nicolaos Matsakis, and Pavel Veselý. Improved approximation guarantees for shortest superstrings using cycle classification by overlap to length ratios. In *Proceedings of the 54th ACM Symposium on Theory of Computing (STOC)*, pages 317–330. ACM, 2022. doi:10.1145/3519935.3520001.
8. Alan M. Frieze and Wojciech Szpankowski. Greedy algorithms for the shortest common superstring that are asymptotically optimal. *Algorithmica*, 21(1):21–36, 1998.
9. Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.
10. Haim Kaplan, Moshe Lewenstein, Nira Shafrir, and Maxim Sviridenko. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. *Journal of the ACM*, 52(4):602–626, 2005. doi:10.1145/1082036.1082041.

11 Haim Kaplan and Nira Shafrir. The greedy algorithm for shortest superstrings. *Inf. Process. Lett.*, 93(1):13–17, 2005. doi:10.1016/j.ipl.2004.09.012.

12 Marek Karpinski and Richard Schmied. Improved inapproximability results for the shortest superstring and related problems. In *Proceedings of the 19th Computing: The Australasian Theory Symposium (CATS)*, pages 27–36, 2013.

13 S. Rao Kosaraju, James K. Park, and Clifford Stein. Long tours and short superstrings. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 166–177, 1994. doi:10.1109/SFCS.1994.365696.

14 Bin Ma. Why greed works for shortest common superstring problem. *Theor. Comput. Sci.*, 410(51):5374–5381, 2009. doi:10.1016/j.tcs.2009.09.014.

15 Marcin Mucha. A tutorial on shortest superstring approximation. https://www.mimuw.edu.pl/~mucha/teaching/aa2008/ss.pdf, 2007. [Accessed 15-June-2023].

16 Marcin Mucha. Lyndon words and short superstrings. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 958–972, 2013. doi:10.1137/1.9781611973105.69.

17 Katarzyna Paluch, Khaled Elbassioni, and Anke van Zuylen. Simpler approximation of the maximum asymmetric traveling salesman problem. In *Proceedings of the 29th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 501–506, 2012. doi:10.4230/LIPIcs.STACS.2012.501.

18 Steven Skiena. *The Algorithm Design Manual, Third Edition.* Texts in Computer Science. Springer, 2020.

19 Ondřej Sladký, Pavel Veselý, and Karel Břinda. Masked superstrings as a unified framework for textual k-mer set representations. *bioRxiv*, 2023. doi:10.1101/2023.02.01.526717.

20 Z. Sweedyk. A 2½-approximation algorithm for shortest superstring. *SIAM J. Comput.*, 29(3):954–986, 1999. doi:10.1137/S0097539796324661.

21 Jorma Tarhio and Esko Ukkonen. A greedy approximation algorithm for constructing shortest common superstrings. *Theor. Comput. Sci.*, 57:131–145, 1988. doi:10.1016/0304-3975(88)90167-3.

22 Shang-Hua Teng and Frances Yao. Approximating shortest superstrings. *SIAM Journal on Computing*, 26(2):410–417, 1997. doi:10.1137/S0097539794286125.

23 Jonathan S. Turner. Approximation algorithms for the shortest common superstring problem. *Inf. Comput.*, 83(1):1–20, 1989. doi:10.1016/0890-5401(89)90044-8.

24 Virginia Vassilevska. Explicit inapproximability bounds for the shortest superstring problem. In *30th International Symposium, MFCS, Gdansk, Poland*, volume 3618 of *Lecture Notes in Computer Science*, pages 793–800. Springer, 2005.

25 Vijay Vazirani. *Approximation algorithms.* Springer, 2001.

A Deriving Approximation Guarantees from Theorem 3

The technical contribution of the paper is proving Theorem 3 that shows an improved inequality for overlaps of cycle-closing edges in terms of the optimal superstring length n and the length w of the optimal cycle cover CC. In the next two subsections, we explain how our improved approximation guarantees follow, using essentially the same arguments (and algorithms) as in previous work.

A.1 The GREEDY Algorithm for SSP

The $|S|^2$ edges of the overlap graph G_{ov} are assumed to be ordered by non-increasing overlap length. The GREEDY algorithm for SSP chooses edges from this order, unless an edge shares an endpoint with an already chosen edge or closes a cycle. The edges corresponding to the latter case are called *bad back edges*. As proven in [3], bad back edges do not intersect each other, forming a laminar family of edges. Each inner-most bad back edge forms a cycle

in the output of GREEDY and each such cycle is called *culprit*. The sum of lengths of all culprit cycles is denoted by w_c and the sum of overlap lengths of the cycle-closing edges of all culprits is denoted by o_c.

Blum et al. have shown the following two inequalities (Section 5 in [3]):

$$|\mathsf{GREEDY}(S)| \leq 2n + o_c - w_c \tag{12}$$

$$o \leq n + 2w \tag{13}$$

Moreover, the application of the GREEDY algorithm for the optimal cycle cover CC on the set of strings comprising the culprit cycles only, outputs the exact same set of culprit cycles (Lemma 15 in [3]). By this and (13) it follows that $o_c \leq n + 2w_c$, which by (12) gives $|\mathsf{GREEDY}(S)| \leq 4n$, completing their proof.

Theorem 3 shows that $o \leq n + \frac{\sqrt{67}-4}{3} w$ which implies that $o_c \leq n + \frac{\sqrt{67}-4}{3} w_c$ using the same syllogism (Lemma 15 in [3]). By this and (12), we have $|\mathsf{GREEDY}(S)| \leq \frac{\sqrt{67}+2}{3} n \approx 3.396 \cdot n$, completing our proof.

A.2 SSP Algorithms Based on Max-ATSP Approximations

Blum et al. proposed the following 4-approximate SSP algorithm, called MGREEDY:

1. Apply GREEDY to find an optimal cycle cover CC.
2. Open all cycle-closing edges in CC to obtain a set of strings called *representatives*.
3. Concatenate the representatives in an arbitrary order.

If instead of concatenating the representatives in the third step, we merge them using a Max-ATSP approximation algorithm (executed on the overlap graph of the representatives), then we will obtain an SSP approximation algorithm which, obviously, cannot perform worse. This is the idea behind the 3-approximate TGREEDY algorithm [3]. The Max-ATSP algorithm utilized as a black-box within TGREEDY is GREEDY, which had been already shown [21, 23] to be a $\frac{1}{2}$-approximate Max-ATSP algorithm for the overlap graphs.

We will need the following theorem from [7], which has already appeared in similar forms in literature (e.g., [3, 4, 15]).

▶ **Theorem 13.** *If MGREEDY is a $(2+\zeta)$-approximate SSP algorithm and there exists a δ-approximate algorithm for Max-ATSP then there exists a $(2+(1-\delta)\cdot\zeta)$-approximate SSP algorithm.*

Showing that $o \leq n + (\sqrt{67}-4)w/3 \approx n + 1.396w$ implies that MGREEDY is a 3.396-approximate SSP algorithm, since $|\mathsf{MGREEDY}(S)| = w + o \leq w + n + (\sqrt{67}-4)w/3 < 3.396n$. Moreover, the currently best Max-ATSP approximation algorithms are $\frac{2}{3}$-approximate, due to Kaplan et al. [10] or due to Paluch et al. [17]. Setting $\delta = \frac{2}{3}$ and $\zeta = (\sqrt{67}-4)/3 \approx 1.396$ in Theorem 13, we obtain an SSP algorithm with approximation guarantee $\frac{\sqrt{67}+14}{9} \approx 2.466$.

Finally, regarding TGREEDY, setting $\delta = \frac{1}{2}$ and $\zeta = (\sqrt{67}-4)/3 \approx 1.396$ in Theorem 13, we improve the approximation guarantee of TGREEDY to $(\sqrt{67}+8)/6 \approx 2.698$, from $(25+\sqrt{57})/12 \approx 2.712$ as shown in [7].

B Dealing with extra large cycles (as in [7])

Let $\overline{S} \subseteq S$ be the subset of strings that belong to all small and large cycles of CC. Observation 5.1 in [7] implies that the optimal cycle cover for $\bar{S}$ (in short $\mathsf{CC}(\overline{S})$) consists of all small and large cycles of the optimal cycle cover for S (for simplicity denoted by $\mathsf{CC}(S) = \mathsf{CC}$), while the optimal cycle cover for $S - \overline{S}$ (in short $\mathsf{CC}(S - \overline{S})$) consists of all extra large cycles of $\mathsf{CC}(S)$.

Let $\hat{w}$ denote the sum of lengths of the (extra large) cycles in $\mathsf{CC}(S - \overline{S})$ and let $\hat{o}$ be the sum of overlap lengths of the cycle-closing edges of the cycles in $\mathsf{CC}(S - \overline{S})$. Similarly, let $\overline{o}$ be the sum of overlap lengths of the cycle-closing edges in $\mathsf{CC}(\overline{S})$ and let $\overline{w}$ be the sum of lengths of the cycles in $\mathsf{CC}(\overline{S})$.

Proving $o \leq n + \beta \cdot w$ for input $\overline{S}$ implies that $\overline{o} \leq |\mathsf{OPT}(\overline{S})| + \beta \cdot \overline{w}$, and assuming this, we show $o \leq n + \beta \cdot w$. Indeed, we take the sum of inequality $\overline{o} \leq |\mathsf{OPT}(\overline{S})| + \beta \cdot \overline{w}$ with inequality $\hat{o} \leq \beta \cdot \hat{w}$ (which holds by the definition of extra large cycles) and obtain:

$$o = \overline{o} + \hat{o} \leq |\mathsf{OPT}(\overline{S})| + \beta \cdot \overline{w} + \beta \cdot \hat{w} = |\mathsf{OPT}(\overline{S})| + \beta \cdot w \leq n + \beta \cdot w$$

where the penultimate step uses $w = \overline{w} + \hat{w}$ and the last inequality uses $|\mathsf{OPT}(\overline{S})| \leq |\mathsf{OPT}(S)| = n$, which follows from $\overline{S} \subseteq S$. Therefore, for proving $o \leq n + \beta \cdot w$, we assume w.l.o.g. that $\mathsf{CC}(S) = \mathsf{CC}$ has no extra large cycle.

C Lemma 11 (slightly modified from [7])

For completeness, we include a proof of Lemma 11. The proof is almost identical to the one in [7] with only very minor changes to make it more general.

We start by stating a corollary, a version of which is already stated in [7] and in slight variations has been known already before (e.g. see Lemma 9 in [3] and Lemma 7 in [15]).

▶ **Corollary 14.** *Let c and c' be any two cycles of CC. Any string h, which is a substring of both $s(c)^\infty$ and $s(c')^\infty$,[1] satisfies $|h| < w(c) + w(c')$.*

This enables us to restate the proof of Lemma 11.

Proof of Lemma 11. We show that $|\mathsf{ov}(e)| > |\mathsf{ov}(f)| + |\mathsf{ov}(f')| - w(c')$, which implies the lemma. If $\min\{|\mathsf{ov}(f)|, |\mathsf{ov}(f')|\} \leq w(c')$, this inequality holds because by using Lemma 7, we get

$$\begin{aligned} |\mathsf{ov}(e)| &\geq w(c) + w(c') \\ &> \max\{|\mathsf{ov}(f)|, |\mathsf{ov}(f')|\} \\ &\geq \max\{|\mathsf{ov}(f)|, |\mathsf{ov}(f')|\} + \min\{|\mathsf{ov}(f)|, |\mathsf{ov}(f')|\} - w(c') \\ &= |\mathsf{ov}(f)| + |\mathsf{ov}(f')| - w(c') \,. \end{aligned}$$

Hence, for the remainder of the proof, we assume that we have $\min\{|\mathsf{ov}(f)|, |\mathsf{ov}(f')|\} > w(c')$.

Now, assume for contradiction that $|\mathsf{ov}(e)| \leq |\mathsf{ov}(f)| + |\mathsf{ov}(f')| - w(c')$. We claim that in this case $\mathsf{ov}(e)$ has a periodicity of length $w(c')$, i.e., $\mathsf{ov}(e)$ is a prefix of x^∞ for some string x with $|x| = w(c')$. To show this, first recall that $|\mathsf{ov}(e)| \geq w(c) + w(c') > \max\{|\mathsf{ov}(f')|, |\mathsf{ov}(f)|\}$ by Lemma 7. Since $\mathsf{ov}(f)$ is a prefix of v and a suffix of v' and since $\mathsf{ov}(e)$ is a prefix of v, the first $|\mathsf{ov}(f)|$ characters of $\mathsf{ov}(e)$ are also a suffix of v', i.e.,

$$\mathsf{ov}(e)[1, |\mathsf{ov}(f)|] = \mathsf{ov}(f) = v'[|v'| - |\mathsf{ov}(f)| + 1, |v'|] \,.$$

Similarly, since $\mathsf{ov}(f')$ is a prefix of u' and a suffix of u and since $\mathsf{ov}(e)$ is a suffix of u, we get that

$$\mathsf{ov}(e)[|\mathsf{ov}(e)| - |\mathsf{ov}(f')| + 1, |\mathsf{ov}(e)|] = \mathsf{ov}(f') = u'[1, |\mathsf{ov}(f')|] \,.$$

[1] The definitions of $s(c)$ and s^∞ appear below Lemma 9.

Observe that for all $1 \leq i \leq |\mathsf{ov}(e)| - w(c')$, a character at position i of $\mathsf{ov}(e)$ must be the same as the character at position $i + w(c')$ of $\mathsf{ov}(e)$. Indeed, if $i + w(c') \leq |\mathsf{ov}(f)|$, this is true as v' has a periodicity of length $w(c')$. If $i > |\mathsf{ov}(e)| - |\mathsf{ov}(f')|$, it is true because u' has a periodicity of length $w(c')$. One of these two cases must apply because otherwise, $i + w(c') > |\mathsf{ov}(f)|$ and $i \leq |\mathsf{ov}(e)| - |\mathsf{ov}(f')|$, which implies $|\mathsf{ov}(f)| - w(c') < i \leq |\mathsf{ov}(e)| - |\mathsf{ov}(f')|$, contradicting our assumption that $|\mathsf{ov}(f')| + |\mathsf{ov}(f)| \geq |\mathsf{ov}(e)| + w(c')$. Hence, $\mathsf{ov}(e)$ has a periodicity of length $w(c')$ (in particular, $\mathsf{period}(\mathsf{ov}(e)) \leq w(c')$).

Next, we show that $\mathsf{ov}(e)$ is a substring of the semi-infinite string $s(c')^\infty$. Because $\mathsf{ov}(e)$ has a periodicity of length $w(c')$ and $s(c')^\infty$ has period $w(c')$, it is sufficient to argue that the first $w(c')$ characters of $\mathsf{ov}(e)$ are a substring of $s(c')^\infty$. This is indeed the case since $\mathsf{ov}(e)[1, |\mathsf{ov}(f)|]$ is a substring of v' which is a substring of $s(c')^\infty$ and we assume that $|\mathsf{ov}(f)| > w(c')$.

Since $\mathsf{ov}(e)$ is a substring of $s(c')^\infty$ as well as of $s(c)^\infty$ (because $\mathsf{ov}(e)$ is a substring of a string that is part of c), Corollary 14 implies $|\mathsf{ov}(e)| < w(c) + w(c')$ which contradicts the assumption of the lemma. ◀

Rapid Mixing for the Hardcore Glauber Dynamics and Other Markov Chains in Bounded-Treewidth Graphs

David Eppstein
Department of Computer Science, University of California, Irvine, CA, USA

Daniel Frishberg
Department of Computer Science and Software Engineering, California Polytechnic State University, San Luis Obispo, CA, USA

Abstract

We give a new rapid mixing result for a natural random walk on the independent sets of a graph G. We show that when G has bounded treewidth, this random walk – known as the *Glauber dynamics* for the *hardcore model* – mixes rapidly for all fixed values of the standard parameter $\lambda > 0$, giving a simple alternative to existing sampling algorithms for these structures. We also show rapid mixing for analogous Markov chains on dominating sets, b-edge covers, b-matchings, maximal independent sets, and maximal b-matchings. (For b-matchings, maximal independent sets, and maximal b-matchings we also require bounded degree.) Our results imply simpler alternatives to known algorithms for the sampling and approximate counting problems in these graphs. We prove our results by applying a divide-and-conquer framework we developed in a previous paper, as an alternative to the *projection-restriction* technique introduced by Jerrum, Son, Tetali, and Vigoda. We extend this prior framework to handle chains for which the application of that framework is not straightforward, strengthening existing results by Dyer, Goldberg, and Jerrum and by Heinrich for the Glauber dynamics on q-colorings of graphs of bounded treewidth and bounded degree.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains

Keywords and phrases Glauber dynamics, mixing time, projection-restriction, multicommodity flow

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.30

Related Version *Full Version*: https://arxiv.org/abs/2111.03898

Acknowledgements The authors wish to acknowledge a number of helpful conversations on this topic with Milena Mihail, Ioannis Panageas, Eric Vigoda, Charlie Carlson, and Zongchen Chen, as well as with Karthik Gajulapalli, Hadi Khodabande, and Pedro Matias.

1 Introduction

The *Glauber dynamics* on independent sets in a graph – motivated in part by modeling systems in statistical physics – is a Markov chain in which one starts at an arbitrary independent set, then repeatedly chooses a vertex at random and, with probability that depends on a fixed parameter $\lambda > 0$, either removes the vertex from the set (if it is in the set), or adds it to the set (if it is not in the set and has no neighbor in the set). This chain, which samples from the *hardcore model* on independent sets, has seen recent rapid mixing results under various conditions. In addition to independent sets, similar dynamics have been studied for a number of other structures – including, for example, q-colorings, matchings, and edge covers (more generally, b-matchings and b-edge covers).

1.1 Our contribution

We prove that the hardcore Glauber dynamics mixes rapidly on graphs of bounded treewidth for all fixed $\lambda > 0$, and that the Glauber dynamics on partial q-colorings (for all $\lambda > 0$) of a graph of bounded treewidth, and on q-colorings of a graph of bounded treewidth and degree, mix rapidly. Marc Heinrich proved the latter result, namely for q-colorings, in a 2020 preprint [10]. Heinrich's result applies to all graphs of bounded treewidth; however, for graphs of bounded treewidth and degree, whose degree is less than quadratic in their treewidth, we improve on Heinrich's upper bound – provided that q is fixed. We also prove that the analogous dynamics on the b-edge covers (when b is bounded) and the dominating sets of a graph of bounded treewidth mix rapidly for all $\lambda > 0$. In a similar vein, we prove that three additional chains – on b-matchings (when $\lambda > 0$), on maximal independent sets, and on maximal b-matchings – mix rapidly in graphs of bounded treewidth and degree.

To prove our results, we apply a framework we introduced in a companion paper [6] that uses the *multicommodity flow* technique (essentially the same as the *canonical paths* technique) for bounding mixing times. (We previously presented this framework in a preprint of the present paper [5].) The framework consists of a set of conditions (which we will define in Section 3.3) that guarantee rapid mixing; these conditions make progress towards unifying prior work on similar Glauber dynamics with prior work on probabilistic graphical models. In that paper [6], we also proved that the *flip walk* on the *k-angulations* of a convex n-point set mixes in time quasipolynomial in n for all fixed $k \geq 3$, although the special case $k = 3$ was known already to mix rapidly [15]. Thus our framework applies beyond graphical models and graph sampling problems.

1.2 Main results

Our main results are the following (see Section 2 for relevant definitions).

▶ **Theorem 1.** *The hardcore Glauber dynamics mixes in time $n^{O(t)}$ on graphs of treewidth t for all fixed $\lambda > 0$.*

▶ **Theorem 2.** *The (unbiased) Glauber dynamics on q-colorings (when $q \geq \Delta + 2$ is fixed) mixes in time $n^{O(t)}$ on graphs of treewidth t and bounded degree when q is fixed. The Glauber dynamics on partial q-colorings (when $q \geq \Delta + 2$ is fixed) mixes in time $n^{O(t)}$ on graphs of treewidth t for all fixed $\lambda > 0$.*

▶ **Theorem 3.** *The Glauber dynamics on b-edge covers mixes in time $n^{O(t^2)}$ on graphs of treewidth t, for all fixed $b \geq 1$ and fixed $\lambda > 0$. The Glauber dynamics on dominating sets mixes in time $n^{O(t)}$ on graphs of treewidth t for all fixed $\lambda > 0$. The Glauber dynamics on b-matchings mixes in time $n^{O(t)}$ on graphs of treewidth t and bounded degree Δ for all fixed $\lambda > 0$ and fixed $b \geq 1$.*

▶ **Theorem 4.** *There exist Markov chains on maximal independent sets and maximal b-matchings, whose stationary distributions are uniform, that mix in time $n^{O(t)}$ on graphs of treewidth t and bounded degree.*

1.3 The framework: recursive flow construction

A *multicommodity flow* in an undirected graph $G = (V, E)$ with n vertices is a set of n^2 flows, one flow for each ordered pair of vertices (s, t), where each flow sends one unit of a commodity from s to t. More precisely, take each (undirected) edge in E and make two directed copies,

one in each direction; let $E^+ = \bigcup_{\{u,v\}\in E}\{(u,v),(v,u)\}$ denote the set of all these directed copies. A multicommodity flow is a collection of functions $f_{st} : E^+ \to \mathbb{R}_{\geq 0}$ such that each f_{st} is a valid flow function, with s (respectively t) having net out flow (respectively in flow) equal to one, and all other vertices having zero net flow. If a multicommodity flow exists in G with small *congestion* – i.e. one in which no edge carries too much flow – then the natural Markov chain whose states are the vertices of G mixes rapidly.

The chains we analyze are natural random walks on a *Glauber graph* $\mathcal{M}(G)$[1] – the graph whose vertices are the structures over which the random walk is performed, and whose edges are the pairs of these structures with symmetric distance equal to one. For example, in the case of independent sets, $\mathcal{M}(G)$ has as its vertex set the collection of all independent sets in G, and as its edge set the collection of all (unordered) pairs of independent sets S, S' in G such that $S = S' \cup \{v\}$ for some $v \in V(G)$. Thus each of these random walks is performed on a graph that may be exponentially large with respect to the size of the input graph. In our previous work [6], we showed that when all of a certain set of conditions hold, we can construct a multicommodity flow in $\mathcal{M}(G)$ with congestion polynomial in $n = |V(G)|$, implying that the walk on $\mathcal{M}(G)$ mixes rapidly. The conditions specify that $\mathcal{M}(G)$ can be partitioned into a small number of induced subgraphs, all of which are approximately the same size, with large numbers of edges between pairs of the subgraphs. The conditions require that each of these induced subgraphs can be decomposed into smaller Glauber graphs that are similar in structure to $\mathcal{M}(G)$. This self similarity allows for the recursive construction of a multicommodity flow, by assembling flows on smaller Glauber graphs together into a flow in $\mathcal{M}(G)$ with small congestion.

1.4 Projection-restriction and prior work on the hardcore model

Prior work on rapid mixing of Markov chains on subset systems includes the special case of matroid polytopes. For this case, recent results [2, 1] have partly solved a 30-year-old conjecture of Mihail and Vazirani [16]. Other prior work uses multicommodity flows (and the essentially equivalent *canonical paths* technique) to obtain polynomial mixing upper bounds on structures of exponential size, including matchings and 0/1 knapsack solutions [17, 9]. Madras and Randall [13] used a decomposition of the hardcore model state space to prove rapid mixing under different conditions. We also decompose the state space, but our approach is different and is more similar to Heinrich's [10] application of the *projection-restriction* technique introduced by Jerrum, Son, Tetali, and Vigoda [11]. This technique involves partitioning the state space of a chain into a collection of sub-state spaces, each of which internally has a good *spectral gap* – a property that implies rapid mixing – and all of which are well connected to one another. Heinrich used the vertex separation properties of bounded-treewidth graphs to obtain an inductive argument: the resulting sub-spaces are themselves *Cartesian products* of chains on smaller graphs, and thus mix rapidly. (See Lemma 16.) We partition the state space recursively using the same vertex separation properties, and indeed for the chains on b-matchings and q-colorings in bounded-treewidth, bounded-degree graphs, combining these properties straightforwardly with the existing spectral projection-restriction machinery of [11] suffices for rapid mixing. The main contribution in this paper is to extend the framework to chains for which this application is not straightforward. That is, we give general conditions for constructing a multicommodity flow in the projection chain with small

[1] The chains on maximal independent sets and maximal b-matchings are not strictly Glauber dynamics, but we will use the same term for the graph, redefining the edge set as pairs connected by the moves we define in the appendix of the full version.

congestion, giving a good spectral gap in the projection chain. One may then apply induction using the spectral machinery of [11] to obtain rapid mixing in the overall chain; alternatively, one can substitute the flow-based machinery from our companion paper [6] for the spectral technique.

In the case of independent sets, Jerrum, Son, Tetali, and Vigoda [11] applied their technique to a special case of the hardcore model, namely regular trees. However, it was not clear how to generalize this application to bounded-treewidth graphs – since showing the spectral gap of the projection chain is sufficiently large is not straightforward. Martinelli, Sinclair, and Weitz [14] showed that the Glauber dynamics on the hardcore model mixes in $O(n \log n)$ time on the complete $\Delta - 1$-ary tree with n nodes, but they did not address general trees. Berger, Kenyon, Mossel, and Peres [3] showed rapid mixing for q-colorings of regular trees with unbounded degree but also did not address general trees. Our first main technical contribution is to show rapid mixing for general bounded-treewidth graphs by introducing the *hierarchical* version of our framework, in which we construct a flow with small congestion in the projection chain; we show that this construction gives rapid mixing for dominating sets, *partial* q-colorings, and b-edge covers in bounded-treewidth graphs. We solve another problem: the technical theorem in [11] as stated requires each of the state spaces in the partition to be a Cartesian product of chains on smaller spaces. For four of our eight chains – those on dominating sets, b-edge covers, maximal b-matchings, and maximal independent sets – the sub-spaces obtained in the decomposition are not a disjoint union of Cartesian products but may each be a union of Cartesian products, or may be mutually intersecting. In some cases, the sub-spaces may even induce disconnected restriction chains. Our second main contribution is to resolve this problem, using the structure of the state spaces of Glauber dynamics as graphs. We discuss this in the appendix of the full version.

1.5 Paper organization

In Section 2, we give relevant background. In Section 3, we use the chain on independent sets to review the "non-hierarchical" version of our framework (the version we gave in our companion paper) – which works for this chain when treewidth and degree are bounded. In the appendix of the full version we apply it to q-colorings and to b-edge covers and b-matchings. To fully prove Theorem 1 and Theorem 3, we need to deal with unbounded-degree graphs – our first main technical contribution. In Section 4, we modify the framework to do so, proving Theorem 1 for $\lambda = 1$. We defer some details to the appdenix of the full version, where we also finish the proof of Theorem 2 for $\lambda = 1$. We prove the general case $\lambda > 0$ of Theorems 1 and 2 in the appendix of the full version. We finish the proofs of Theorems 3 and 4 in the appendix of the full version: applying the framework to the relevant chains requires a further refinement of the framework. In all of the above, we prove rapid mixing but defer derivation of specific upper bounds to the appendix of the full version.

2 Preliminaries

2.1 Glauber dynamics

▶ **Definition 5.** *The* hardcore Glauber dynamics *on a graph G is the following chain, defined with respect to a fixed real parameter $\lambda > 0$:*

1. *Let X_0 be an arbitrary independent set in G.*
2. *For $t \geq 0$, select a vertex $v \in V(G)$ uniformly at random.*
3. *If $v \notin X_t$ and $X_t \cup \{v\}$ is not a valid independent set, do nothing.*

4. *Otherwise:*
   Let $X_{t+1} = X_t \cup \{v\}$ *with probability* $\lambda/(\lambda+1)$.
   Let $X_{t+1} = X_t \setminus \{v\}$ *with probability* $1/(\lambda+1)$.

Graph-theoretically, the Glauber dynamics is defined as follows: let the *independent set Glauber graph* $\mathcal{M}_{\mathrm{IS}}(G)$ denote the graph whose vertices are identified with the independent sets of a given graph G, and whose edges are the pairs of independent sets whose symmetric difference is one. The hardcore Glauber dynamics is a Markov chain, parameterized by $\lambda > 0$, with state space $\Omega = V(\mathcal{M}_{\mathrm{IS}}(G))$ and probability matrix P, where for $S, S' \in V(\mathcal{M}(G))$ with $S \neq S'$, $P(S, S') = \lambda/(\Delta_{\mathcal{M}}(\lambda+1))$ when $|S' \setminus S| = 1$, and $P(S, S') = 1/(\Delta_{\mathcal{M}}(\lambda+1))$ when $|S \setminus S'| = 1$. If $S = S'$, then $P(S, S') = 1 - \sum_{S'' \neq S} P(S, S'')$. (Here $\Delta_{\mathcal{M}}$ is the maximum degree of the Glauber graph – i.e. the maximum number of neighboring states that a state S can have.) The Glauber graph has vertex set Ω and adjacency matrix P – up to the addition of self loops and normalization by degree. (When $\lambda \neq 1$ this graph can still be augmented with suitable weights so that the walk on the graph is the Glauber dynamics.)

2.2 Mixing time

To generate, approximately uniformly at random, an object of a given class – such as an independent set in a given graph – one can conduct a random walk on a graph whose vertices are the objects of interest, and whose edges represent local moves between the objects (or states). It is known that under certain mild conditions satisfied by as all our chains (see the appendix of the full version), the walk converges to the uniform distribution in the limit. The rate of convergence is important: in the case of subset systems such as those we consider, the walk takes place over an exponentially large number of subsets defined over an underlying set of size n. If the convergence, or *mixing time*, of the walk is polynomial in n, then the random walk is said to be *rapidly mixing*. The mixing time is denoted $\tau(\varepsilon)$, where ε denotes the desired precision of convergence to the uniform distribution, and the value of τ at ε is the minimum number of steps in the random walk before convergence is guaranteed. Convergence is measured via the *total variation distance* [19] between the distribution over states induced by the walk at a given time step, and the uniform distribution. One can obtain non-uniform *stationary* distributions by weighting the graph – see the appendix of the full version. See Levin, Peres, and Wilmer [12] for a comprehensive treatment of rapid mixing.

A Markov chain, given a starting state $S \in \Omega$, induces a probability distribution π_t at each time step t. The Glauber dynamics is known, regardless of starting state, to converge in the limit to a *stationary* distribution $\pi^*(S) = \lambda^{|S|}/Z(\mathcal{M}(G))$, where the term $Z(\mathcal{M}(G))$ is simply a normalizing value. When λ is unspecified, assume $\lambda = 1$ (the uniform case). The *mixing time* is defined as follows:

Given an arbitrary $\varepsilon > 0$, the *mixing time*, $\tau(\varepsilon)$, of a Markov chain with state space Ω and stationary distribution π^* is the minimum time t such that, regardless of starting state, we always have $\frac{1}{2}\sum_{S \in \Omega} |\pi(S) - \pi^*(S)| < \varepsilon$. Suppose the chain belongs to a family of Markov chains, the size of whose state space is parameterized by some value n. Here, $|\Omega|$ may be exponential in n. If $\tau(\varepsilon)$ is bounded by a polynomial function in $\log(1/\varepsilon)$ and in n, the chain is said to be *rapidly mixing*. It is common to omit the parameter ε and assume $\varepsilon = 1/4$.

2.3 Treewidth and vertex separators

▶ **Definition 6** ([18]). *A* tree decomposition *of a graph* $G = (V, E)$ *is a collection of sets* $\{X_i\}, i = 1, \ldots, k$, *called* bags, *together with a tree* T, *whose nodes are identified with the bags* $\{X_i\}$, *such that all of the following hold:*

1. *Every vertex in V lies in some bag, i.e. $\bigcup_{i=1}^{k} X_i = V$.*
2. *For every $(u, v) \in E$, the vertices u and v belong to at least one bag X_i together, i.e. for some i, $u \in X_i$ and $v \in X_i$.*
3. *The collection of all bags containing any given vertex $v \in V$, i.e. $\{X_i \mid v \in X_i\}$ forms a (connected) subtree of T.*

▶ **Definition 7** ([18]). *The* width *of a tree decomposition is one less than the size of the largest bag in the decomposition. The* treewidth *of a graph G is the minimum t such that a tree decomposition of G exists with width t.*

Intuitively, treewidth measures how far away a graph is from being a tree. For example, trees have treewidth one; a graph consisting of a single cycle of size at least three has treewidth two. Treewidth is of interest in large part because many NP-hard problems become tractable on graphs of bounded treewidth. For a full definition of treewidth and a survey of this phenomenon, known as *fixed-parameter tractability*, see [4].

For our purposes, treewidth is of interest due to its relationship to *vertex separators*: a vertex set $X \subseteq V$ in a graph $G = (V, E)$ is called a *vertex separator* if the deletion of X from V leaves the induced subgraph on the remaining vertices disconnected. Say that X is a *balanced separator* if deleting X partitions V into mutually disconnected subsets $A \cup B = V \setminus X$ such that $|V|/3 \leq |A| \leq |B| \leq 2|V|/3$. A graph G is *recursively s-separable* [7] if either (i) $|V(G)| \leq 1$, or (ii) G has a balanced separator X with $|X| \leq s$ and, after deleting X, the resulting subsets A and B induce subgraphs of G that are each recursively s-separable.

The following is known and easy to prove [7]:

▶ **Lemma 8.** *Every graph with treewidth $t \geq 1$ is recursively s-separable for all $s \geq t + 1$.*

3 $\lambda = 1$: Bounded treewidth and degree

To build up to the proof of Theorem 1, we first show a weaker result: that the unifrom hardcore Glauber dynamics mixes rapidly in graphs of bounded treewidth and degree. Fully proving Theorem 1, even in the unbiased case, requires the non-hierarchical framework. The main technical lemma in this section, Lemma 17, comes from our companion paper. Our contribution in this paper is the application to independent sets in graphs of bounded treewidth and degree – which we strengthen to graphs of bounded treewidth in Section 4.

The following is necessary for the Glauber dynamics to sample correctly:

▶ **Lemma 9.** *The independent set Glauber graph is connected.*

Proof. Consider the empty independent set $\emptyset$. Every independent set $S \in V(\mathcal{M}_{\mathrm{IS}}(G))$ has a path of length $|S|$ to $\emptyset$, formed by removing each vertex in S in arbitrary order. ◀

3.1 Partitioning the vertices of $\mathcal{M}_{\mathrm{IS}}(G)$ into classes

The vertices of the Glauber graph $\mathcal{M}_{\mathrm{IS}}(G)$ are subsets of the vertices of an underlying graph G. When G has bounded treewidth, we can choose a small separator X that partitions $V(G) \setminus X$ into two mutually disconnected vertex subsets, A and B, each of which has at most $2|V(G)|/3$ vertices. Consider the problem of sampling an independent set S from G. Given a separator X for G, partition the independent sets in G into equivalence classes as follows:

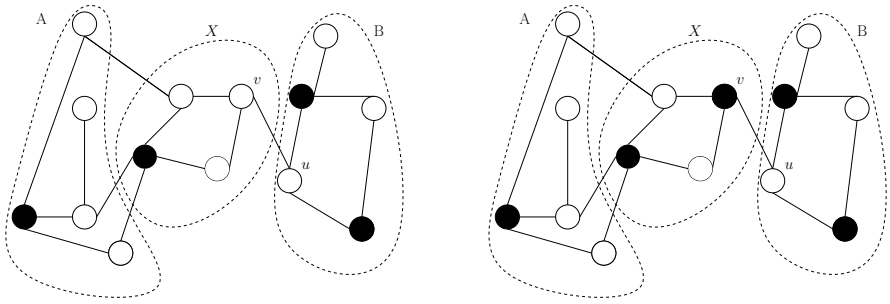


Figure 1 Two independent sets in a graph G: S (left) and S' (right), belonging to distinct classes. S and S' differ by a move, with the separator X inducing the classes to which the sets belong. S' results from adding v to S. $|S'| < |S|$, since S' excludes those independent sets that contain u.

▶ **Definition 10.** *Let $G = (V, E)$ be a graph. Let $\mathcal{M}_{\mathrm{IS}}(G)$ be the independent set Glauber graph we have defined. Let $X \subseteq V$ be a vertex separator for G. Let $\mathcal{S}_{\mathrm{IS}}(G)$ be the set of equivalence classes of $V(\mathcal{M}_{\mathrm{IS}}(G))$ in which two independent sets S and S' are in the same class if $S \cap X = S' \cap X$. Let $T = S \cap X$, and call the corresponding class $\mathcal{C}_{\mathrm{IS}}(T)$.*

(Technically X is also a parameter for $\mathcal{S}_{\mathrm{IS}}(G)$ and $\mathcal{C}_{\mathrm{IS}}(T)$, but we omit it for ease of notation.)

See Figure 1 for an example of a partitioning.

The *Cartesian product* of two graphs H and J is the graph whose vertex set is $V(H) \times V(J)$ and whose edges are the pairs $((h_1, j_1), (h_2, j_2))$ such that either $h_1 = h_2$ and $(j_1, j_2) \in E(J)$ or else $(h_1, h_2) \in E(H)$ and $j_1 = j_2$.

Let A and B be the mutually disconnected vertex subsets into which the removal of X partitions $V(G) \setminus X$. Given a fixed independent subset $T \subseteq X$, identify the independent sets in $\mathcal{C}_{\mathrm{IS}}(T)$ with the pairs of the form (S_A, S_B), where S_A is an independent set in $A \setminus N_A(T)$, and S_B is an independent set in $B \setminus N_B(T)$, where $N_A(T)$ and $N_B(T)$ denote the union of the neighborhoods of vertices in T, in A and B respectively. That is, identify each independent set in $\mathcal{C}_{\mathrm{IS}}(T)$ with a pair of an independent set in A that avoids neighbors of vertices in T, and a similar independent set in B. Consider the two Glauber graphs $\mathcal{M}_{\mathrm{IS}}(A \setminus N_A(T))$ and $\mathcal{M}_{\mathrm{IS}}(B \setminus N_B(T))$, whose vertices are respectively the independent sets in $G[A \setminus N_A(T)]$, and those in $G[B \setminus N_B(T)]$. If two independent sets $S = (S_A, S_B)$ and $S' = (S'_A, S'_B)$ belong to the same class, then a *move* (traversal of an edge in the Glauber graph) exists between S and S' in $\mathcal{M}_{\mathrm{IS}}(G)$ precisely when a move exists between the restrictions of S and S' to either $\mathcal{M}_{\mathrm{IS}}(A \setminus N_A(T))$ or $\mathcal{M}_{\mathrm{IS}}(B \setminus N_B(T))$ (but not both). Therefore, each class induces, in $\mathcal{M}_{\mathrm{IS}}(G)$, a subgraph that is isomorphic to a Cartesian product of two smaller Glauber graphs:

▶ **Lemma 11.** *Given a graph G and a vertex separator X that partitions $V(G)$ into subgraphs A and B, for every class $T \in \mathcal{S}_{\mathrm{IS}}(G)$, $\mathcal{C}_{\mathrm{IS}}(T) \cong \mathcal{M}_{\mathrm{IS}}(A \setminus N_A(T)) \square \mathcal{M}_{\mathrm{IS}}(B \setminus N_B(T))$.*

(Here by the symbol $\cong$ we denote isomorphism, and we identify the class $\mathcal{C}_{\mathrm{IS}}(T)$ with the subgraph it induces in $\mathcal{M}_{\mathrm{IS}}(G)$.)

3.2 Rapid mixing for the hardcore Glauber dynamics when G has bounded treewidth and degree

As described in Section 3.1, we use a small vertex separator X in G to give a decomposition of $\mathcal{M}_{\mathrm{IS}}(G)$ into subgraphs, each of which has a Cartesian product structure – in which both factor graphs in the product are themselves Glauber graphs. Since Cartesian products preserve flow congestion upper bounds (see Lemma 16), this decomposition provides a crucial inductive structure. We analyze this structure in this section.

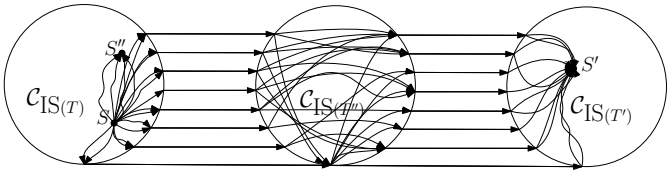


Figure 2 A schematic view of three classes in the independent set Glauber graph $\mathcal{M}_{\mathrm{IS}}(G)$. The large circles denote classes under the partition described in Section 3.1. The curved arrows illustrate the construction of a flow in $\mathcal{M}_{\mathrm{IS}}(G)$ from an independent set $S \in \mathcal{C}_{\mathrm{IS}}(T)$ to another independent set $S'' \in \mathcal{C}_{\mathrm{IS}}(T)$ – and also to an independent set $S' \in \mathcal{C}_{\mathrm{IS}}(T')$. Here, $\mathcal{C}_{\mathrm{IS}}(T)$ and $\mathcal{C}_{\mathrm{IS}}(T'')$ are adjacent classes in $\mathcal{M}_{\mathrm{IS}}(G)$, connected by a large number of edges, and similarly $\mathcal{C}_{\mathrm{IS}}(T')$ and $\mathcal{C}_{\mathrm{IS}}(T'')$ are adjacent. In Section 3.2 we formalize this flow.

▶ **Lemma 12.** *Let G be a graph with bounded treewidth t and bounded degree Δ, let $\mathcal{M}_{\mathrm{IS}}(G)$ be as we have defined, and let $\mathcal{S}_{\mathrm{IS}}(G)$ be as in Definition 10 with respect to a small balanced separator X with $|X| \leq t+1$. Then:*

1. *The number of classes in $\mathcal{S}_{\mathrm{IS}}(G)$ is $O(1)$.*
2. *For every pair of classes $\mathcal{C}_{\mathrm{IS}}(T), \mathcal{C}_{\mathrm{IS}}(T') \in \mathcal{S}_{\mathrm{IS}}(G)$, $|\mathcal{C}_{\mathrm{IS}}(T)| = \Theta(1)|\mathcal{C}_{\mathrm{IS}}(T')|$.*
3. *Let $\mathcal{C}_{\mathrm{IS}}(T), \mathcal{C}_{\mathrm{IS}}(T') \in \mathcal{S}_{\mathrm{IS}}(G)$ be two classes. No independent set $S \in \mathcal{C}_{\mathrm{IS}}(T)$ has more than one move to an independent set $S' \in \mathcal{C}_{\mathrm{IS}}(T')$.*
4. *Let $\mathcal{C}_{\mathrm{IS}}(T), \mathcal{C}_{\mathrm{IS}}(T') \in \mathcal{S}_{\mathrm{IS}}(G)$ be two classes. Suppose there exists at least one move between an independent set in $\mathcal{C}_{\mathrm{IS}}(T)$ and an independent set in $\mathcal{C}_{\mathrm{IS}}(T')$. Then there exist at least $\Omega(1)|\mathcal{C}_{\mathrm{IS}}(T)|$ moves between independent sets in $\mathcal{C}_{\mathrm{IS}}(T)$ and independent sets in $\mathcal{C}_{\mathrm{IS}}(T')$.*

Proof. Claim 1 follows from the fact that $|\mathcal{S}_{\mathrm{IS}}(G)| \leq 2^{|X|} \leq 2^{t+1} = O(1)$, where the first inequality is true because each class is identified with a subset of the vertices in X. The proofs of claims 2 through 4 are in the appendix of the full version. ◀

We will use Lemma 12 to prove the following, applying the framework from our previous paper, in Section 3.3:

▶ **Lemma 13.** *Given a graph G with bounded treewidth and degree, the natural random walk on the independent set Glauber graph $\mathcal{M}_{\mathrm{IS}}(G)$ has mixing time $\tau(\varepsilon) = O(n^c \log 1/\varepsilon)$, where $c = O(1)$.*

To prove Theorem 1, however, we need to get rid of the assumption that degree is bounded. We address this issue in Section 4.

3.3 Abstraction into framework conditions

The observations in Lemma 12 correspond to a set of conditions we gave in our previous work [6]. These conditions are, given a connected graph $\mathcal{M}(G)$, on some set of combinatorial structures over an underlying graph G with n vertices:

1. The vertices of $\mathcal{M}(G)$ can be partitioned into a set $\mathcal{S}$ of classes, where $|\mathcal{S}| = O(1)$.
2. The ratio of the sizes of any two classes in $\mathcal{S}$ is $\Theta(1)$.
3. Given two classes $\mathcal{C}(T), \mathcal{C}(T') \in \mathcal{S}$, no vertex in $\mathcal{C}(T)$ has more than $O(1)$ edges to vertices in $\mathcal{C}(T')$.
4. For every pair of classes that share at least one edge, the number of edges between the two classes is $\Theta(1)$ times the size of each of the two classes.
5. Each class in $\mathcal{S}$ is the Cartesian product of two graphs $\mathcal{M}(G_1)$ and $\mathcal{M}(G_2)$, each of which can be recursively partitioned in the same way as $\mathcal{M}(G)$.
6. The recursive partitioning mentioned in Condition 5 reaches the base case (graphs with one or zero vertices) in $O(\log n)$ steps.

Conditions 1 through 4 correspond respectively to Lemma 12; Condition 5 corresponds to Lemma 11. Condition 6 corresponds to the observation at the end of the proof sketch of Lemma 13.

We introduce some facts that we previously used to prove that these conditions suffice for rapid mixing, via *expansion*, then review a sketch of the proof; we will build on these techniques in Section 4 (our main contribution) and in the appendices.

The *edge expansion* (or simply the *expansion*) of a graph $G = (V, E)$ is the quantity $h(G) := \min_{S \subseteq V : |S| \leq |V|/2} \frac{|\{(u,v) \in E | u \in S, v \notin S\}|}{|S|}$, i.e. the minimum quotient of the number of edges in the cut by the number of vertices on the smaller side of the cut. The *vertex expansion* is the quantity $h(G) := \min_{S \subseteq V : |S| \leq |V|/2} \frac{|\{v \in V \setminus S | \exists u \in S, (u,v) \in E\}|}{|S|}$, i.e. the minimum quotient of the number of neighbors of a set S with $|S| \leq |V|/2$ that are not in S.

Mixing can be bounded from above via a lower bound on expansion [19] when the degree of a Glauber graph is small (linear in the case of our chains):

▶ **Lemma 14.** *Given a graph $\mathcal{M} = (\mathcal{V}, \mathcal{E})$, consider the Markov chain whose state space is $\mathcal{V}$ and whose transitions are of the form $P(x,x) = 1/2, P(x,y) = 0 \forall (x,y) \notin \mathcal{E}, P(x,y) = 1/(2\Delta) \forall (x,y) \in \mathcal{E}$, where Δ is the maximum degree of $\mathcal{M}$. The mixing time of this Markov chain is at most*

$$\tau(\varepsilon) = O\left(\frac{\Delta^2}{(h(\mathcal{M}))^2} \cdot \ln \frac{|\mathcal{V}|}{\varepsilon}\right).$$

Expansion, in turn, can be bounded from below via an upper bound on the *congestion* of a multicommodity flow. Given a multicommodity flow $f = \{f_{st} | s, t \in V \times V\}$ in a graph $G = (V, E)$, define the congestion of f as the quantity $\rho = \max_{(u,v) \in E^+} \sum_{s,t \in V \times V} f_{st}(u,v)$, i.e. the maximum amount of flow sent across an edge.

▶ **Lemma 15** ([19])**.** *For every graph $G = (V, E)$ and for every flow function f defined over G and having congestion ρ, $h(G) \geq 1/(2\rho)$.*

▶ **Lemma 16.** *Given graphs H and J, let G be the Cartesian product $H \square J$. Suppose multicommodity flows exist in H and J with congestion at most ρ_H and ρ_J respectively. Then there exists a multicommodity flow in G with congestion at most $\max\{\rho_H, \rho_J\}$.*

We proved Lemma 16 in [6], although an analogous result for expansion is known [8]. We also proved the following in [6]. We review the lemma and give a modified proof sketch here, in terms more intuitive for the chains we are analyzing in this paper. We will modify the technique in Section 4.

▶ **Lemma 17.** *Given a graph $\mathcal{M}(G)$ satisfying the conditions in Section 3.3, the expansion of $\mathcal{M}(G)$ is $\Omega(1/n^c)$, where $c = O(1)$.*

Proof Sketch. Partition $\mathcal{M}(G)$ into classes as in Definition 10. By Lemma 11, each class $\mathcal{C}(T) \in \mathcal{S}(G)$ is isomorphic to the Cartesian product $\mathcal{M}(A \setminus N_A(T)) \square \mathcal{M}(B \setminus N_B(T))$. We make an inductive argument, in which the inductive hypothesis assumes that for each such Cartesian product, the graphs $\mathcal{M}(A \setminus N_A(T))$ and $\mathcal{M}(B \setminus N_B(T))$ have multicommodity flows f_A and f_B with congestion $\rho_A \leq c^{\log |V(G)| - 1}, \rho_B \leq c^{\log |V(G)| - 1}$ respectively, for some constant c. By Lemma 16, $\mathcal{C}(T)$ then has a flow f_T with congestion $\rho_T \leq c^{\log |V(G)| - 1}$. The inductive step is to combine the f_T flows for all of the classes, giving a flow f in $\mathcal{M}(G)$ with small congestion. We need to route flow between every $S, S' \in V(\mathcal{M}(G))$. If S and S' belong to the same class $\mathcal{C}(T)$, simply use the same flow that S uses to send its unit to S' in f_T. If $S \in \mathcal{C}(T)$ and $S' \in \mathcal{C}(T') \neq \mathcal{C}(T)$ belong to different classes, we find a sequence

of intermediate classes through which to route flow from $\mathcal{C}(T)$ to $\mathcal{C}(T')$. See Figure 2. We specified in our companion paper [6] how to route the flow through each intermediate class so that the congestion across each edges between a pair of classes is at most $O(1)$, then made use of the existing flows within each class guaranteed by the inductive hypothesis to bound the resulting amount of flow within each class $\mathcal{C}(T)$. We showed that the latter is at most $O(1) \cdot \rho_T$, giving overall congestion $O(1)^l$, where l is the number of induction levels. Since X is a balanced separator we have $l = O(\log n)$; the lemma now follows from Lemma 14. ◀

The full proof of Lemma 17 is in our companion paper [6]. We will use the phrase "non-hierarchical framework" to describe this set of conditions – which apply to the chains we study when the underlying graph G has bounded treewidth and degree. Although Jerrum, Son, Tetali, and Vigoda [11] did not consider bounded-treewidth graphs generally, these conditions do allow their projection-restriction technique to be applied. In effect, Lemma 17 and its proof, which we gave in our previous work, characterize a sufficient set of conditions for applying Jerrum, Son, Tetali, and Vigoda's technique: specifically, one can, instead of routing flow internally through each intermediate class, simply treat the construction above as a flow in the projection graph, concluding that the projection graph has a good spectral gap – then apply [11]. The first main technical contribution of this paper is in Section 4, in which we give an alternative set of conditions – which we will call our "hierarchical framework" – that allows us to handle underlying graphs of unbounded degree (though treewidth still must be bounded), and to handle chains other than the hardcore model. This will allow us to complete the proofs of Theorems 1, 2, and 3.

4 $\lambda = 1$: Unbounded degree

4.1 Hierarchical framework

We now sketch "hierarchical" framework conditions that guarantee rapid mixing in the case of unbounded degree (when treewidth is bounded). Several of the chains we consider satisfy these conditions so long as the treewidth of the underlying graph is bounded. This is the first main technical contribution in this paper. In the original framework, we assumed that the classes were approximately the same size. Although all of the graphs to which we apply this hierarchical framework satisfy this condition in graphs with bounded treewidth and degree, this is not the case when the degree is unbounded. Fortunately, in the case of independent sets, partial q-colorings, dominating sets, and b-edge covers, we solve this problem with some modifications to the framework.

4.2 Independent sets

In the proof of Lemma 17, the assumption that the classes were approximately the same size allowed every class $\mathcal{C}_{\text{IS}}(T)$ to route flow for all pairs of vertices without being too congested, because $\mathcal{C}_{\text{IS}}(T)$ is sufficiently large. Once we discard this assumption, we need to specify explicitly the path through which a given $\mathcal{C}_{\text{IS}}(T)$ routes flow to each $\mathcal{C}_{\text{IS}}(T')$. We construct a flow in which for every such $\mathcal{C}_{\text{IS}}(T), \mathcal{C}_{\text{IS}}(T')$, every intermediate class $\mathcal{C}_{\text{IS}}(T'')$ that handles flow between sets $S \in \mathcal{C}_{\text{IS}}(T)$ and $S' \in \mathcal{C}_{\text{IS}}(T')$ is larger than one of $\mathcal{C}_{\text{IS}}(T)$ or $\mathcal{C}_{\text{IS}}(T')$. We then bound the number of pairs of sets, relative to $|\mathcal{C}_{\text{IS}}(T'')|$, for which $\mathcal{C}_{\text{IS}}(T'')$ carries flow.

To accomplish this, we observe that for any $\mathcal{C}_{\text{IS}}(T), \mathcal{C}_{\text{IS}}(T')$ such that there exists *one* move between an independent set in $\mathcal{C}_{\text{IS}}(T')$ and an independent set in $\mathcal{C}_{\text{IS}}(T)$, either *every* independent set in $\mathcal{C}_{\text{IS}}(T')$ has a move to some independent set in $\mathcal{C}_{\text{IS}}(T)$, or vice versa. This move consists of dropping some vertex v from $T' \subseteq X$ to obtain T, i.e. $T = T' \setminus \{v\}$.

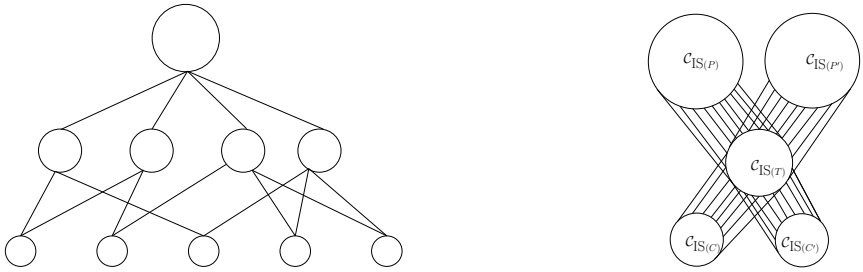


Figure 3 Left: a schematic representation of the classes in the independent set Glauber graph and edges between them when the degree is unbounded. Right: a class $\mathcal{C}_{\mathrm{IS}}(T)$, with two parents, $\mathcal{C}_{\mathrm{IS}}(P)$ and $\mathcal{C}_{\mathrm{IS}}(P')$, and two children, $\mathcal{C}_{\mathrm{IS}}(C)$ and $\mathcal{C}_{\mathrm{IS}}(C')$. (Classes with larger cardinality are drawn larger.) The parallel edges depict the fact that a child class always has every one of its vertices adjacent to a vertex in a given parent class, and that the edges between any given pair of classes are vertex-disjoint.

We call $\mathcal{C}_{\mathrm{IS}}(T)$ a *parent* of $\mathcal{C}_{\mathrm{IS}}(T')$, and $\mathcal{C}_{\mathrm{IS}}(T')$ a *child* of $\mathcal{C}_{\mathrm{IS}}(T)$. See Figure 3. Since the set of edges connecting vertices in $\mathcal{C}_{\mathrm{IS}}(T)$ with vertices in $\mathcal{C}_{\mathrm{IS}}(T')$ forms a matching, this implies that $|\mathcal{C}_{\mathrm{IS}}(T)| \geq |\mathcal{C}_{\mathrm{IS}}(T')|$. In fact, whenever $T \subseteq T'$, we have $|\mathcal{C}_{\mathrm{IS}}(T)| \geq |\mathcal{C}_{\mathrm{IS}}(T')|$. We route flow between any pair of classes $\mathcal{C}_{\mathrm{IS}}(T)$ and $\mathcal{C}_{\mathrm{IS}}(T')$ along a path through a "least common ancestor". Since for every class $\mathcal{C}_{\mathrm{IS}}(T'')$ on this path, either $|\mathcal{C}_{\mathrm{IS}}(T'')| \geq |\mathcal{C}_{\mathrm{IS}}(T)|$ or $|\mathcal{C}_{\mathrm{IS}}(T'')| \geq |\mathcal{C}_{\mathrm{IS}}(T')|$, we obtain a bound on congestion that we make precise in the appendix of the full version.

In the proof sketch of Lemma 17 (Section 3.2), for every pair $S \in \mathcal{C}(T), S' \in \mathcal{C}(T') \neq \mathcal{C}(T)$, we found a sequence of classes $\mathcal{C}(T) = \mathcal{C}(T_1), \mathcal{C}(T_2), \ldots, \mathcal{C}(T_{k-1}), \mathcal{C}(T_k) = \mathcal{C}(T')$ through which to route the $S - S'$ flow. As discussed in Section 4, when the degree is unbounded, the classes are no longer nearly the same size – so if this sequence is chosen carelessly, some $\mathcal{C}(T_i)$ may carry flow for too many $S - S'$ pairs. We therefore choose the sequences carefully: the parent-child relationships induce a partial order $\prec$ on the classes with a unique maximal element, where $\mathcal{C}(T) \succ \mathcal{C}(T')$ implies $|\mathcal{C}(T)| \geq |\mathcal{C}(T')|$. We choose our sequence so that for some i with $1 \leq i \leq k$, $|\mathcal{C}(T_1)| \leq |\mathcal{C}(T_2)| \leq \cdots \leq |\mathcal{C}(T_i)| \geq |\mathcal{C}(T_{i+1})| \geq \cdots |\mathcal{C}(T_{k-1})| \geq |\mathcal{C}(T_k)|$.

4.3 Hierarchical Framework Conditions

The conditions are as follows. Conditions 2 through 4 are new and concern the partial order described above; Condition 1 and Conditions 5 through 7 are as in Section 3.3.

1. The vertices of $\mathcal{M}(G)$ can be partitioned into a set $\mathcal{S}$ of classes, where $|\mathcal{S}| = O(1)$.
2. There exists a partial order $\prec$ on the classes in $\mathcal{S}$, such that whenever $\mathcal{C}(T), \mathcal{C}(T') \in \mathcal{S}$ and $\mathcal{C}(T) \succ \mathcal{C}(T')$, we have $|\mathcal{C}(T)| \geq |\mathcal{C}(T')|$.
3. The partial order $\prec$ has a unique maximal element.
4. Whenever an edge exists between vertices in $\mathcal{C}(T)$ and $\mathcal{C}(T')$ with $\mathcal{C}(T) \succ \mathcal{C}(T')$, the number of such edges is $|\mathcal{C}(T')|$.
5. For every pair of classes $\mathcal{C}(T)$ and $\mathcal{C}(T')$ that share an edge, the maximum degree, in $\mathcal{C}(T)$, of a vertex in $\mathcal{C}(T')$, is $O(1)$, and the maximum degree, in $\mathcal{C}(T')$, of a vertex in $\mathcal{C}(T)$, is $O(1)$.
6. Each class in $\mathcal{S}$ is the Cartesian product of two graphs $\mathcal{M}(G_1)$ and $\mathcal{M}(G_2)$, each of which can be recursively partitioned in the same way as $\mathcal{M}(G)$.
7. The recursive partitioning mentioned in Condition 6 reaches the base case (graphs with one or zero vertices) in $O(\log n)$ levels of recursion.

▶ **Lemma 18.** *Given a graph $\mathcal{M}(G)$ satisfying the conditions in Section 4.3, the expansion of $\mathcal{M}(G)$ is $\Omega(1/n^c)$, where $c = O(1)$.*

We defer the proof of Lemma 18 to the appendix of the full version.

References

1. Nima Anari, Kuikui Liu, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials II: High-dimensional walks and an FPRAS for counting bases of a matroid. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019)*, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316385.
2. Nima Anari, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials, entropy, and a deterministic approximation algorithm for counting bases of matroids. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 35–46, 2018. doi:10.1109/FOCS.2018.00013.
3. Noam Berger, Claire Kenyon, Elchanan Mossel, and Yuval Peres. Glauber dynamics on trees and hyperbolic graphs. *Probability Theory and Related Fields*, 131:311–340, 2005. doi:s00440-004-0369-4.
4. Hans L. Bodlaender. Fixed-parameter tractability of treewidth and pathwidth. In *The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 196–227. Springer, 2012. doi:10.1007/978-3-642-30891-8_12.
5. David Eppstein and Daniel Frishberg. Rapid mixing of the hardcore Glauber dynamics and other Markov chains in bounded-treewidth graphs. *arXiv preprint*, 2021. doi:10.48550/arXiv.2111.03898.
6. David Eppstein and Daniel Frishberg. Improved Mixing for the Convex Polygon Triangulation Flip Walk. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261, pages 56:1–56:17, 2023. doi:10.4230/LIPIcs.ICALP.2023.56.
7. Jeff Erickson. Computational topology: Treewidth. Lecture Notes, 2009. URL: http://jeffe.cs.illinois.edu/teaching/comptop/2009/notes/treewidth.pdf.
8. F. Graham and P. Tetali. Isoperimetric inequalities for cartesian products of graphs. *Comb. Probab. Comput.*, 7:141–148, 1998.
9. Venkatesan Guruswami. Rapidly mixing markov chains: A comparison of techniques (a survey). *arXiv*, 2016. arXiv:1603.01512.
10. Marc Heinrich. Glauber dynamics for colourings of chordal graphs and graphs of bounded treewidth, 2020. arXiv:2010.16158.
11. Mark Jerrum, Jung-Bae Son, Prasad Tetali, and Eric Vigoda. Elementary bounds on Poincaré and log-Sobolev constants for decomposable Markov chains. *The Annals of Applied Probability*, 14(4):1741–1765, 2004. URL: http://www.jstor.org/stable/4140446.
12. David A Levin, Elizabeth Wilmer, and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Society, 2009. URL: https://bookstore.ams.org/mbk-58.
13. Neal Madras and Dana Randall. Markov chain decomposition for convergence rate analysis. *The Annals of Applied Probability*, 12(2):581–606, 2002. doi:10.1214/aoap/1026915617.
14. Fabio Martinelli, Alistair Sinclair, and Dror Weitz. Fast mixing for independent sets, colorings, and other models on trees. *Random Structures & Algorithms*, 31(2):134–172, 2007. doi:10.1002/rsa.20132.
15. Lisa McShine and P. Tetali. On the mixing time of the triangulation walk and other catalan structures. In *Randomization Methods in Algorithm Design*, 1997.
16. Milena Mihail and Umesh Vazirani. On the expansion of 0-1 polytopes. *Journal of Combinatorial Theory, Series B*, 1989.
17. Ben Morris and Alistair Sinclair. Random walks on truncated cubes and sampling 0-1 knapsack solutions. *SIAM journal on computing*, 34(1):195–226, 2004.

18 Neil Robertson and P.D Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

19 Alistair Sinclair. Improved bounds for mixing rates of Markov chains and multicommodity flow. *Combinatorics, Probability and Computing*, 1(4):351–370, 1992. doi:10.1017/S0963548300000390.

Matching Cuts in Graphs of High Girth and H-Free Graphs

Carl Feghali
University of Lyon, EnsL, CNRS, LIP, F-69342, Lyon Cedex 07, France

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 (Perfect) Matching Cut problem is to decide if a connected graph has a (perfect) matching that is also an edge cut. The Disconnected Perfect Matching problem is to decide if a connected graph has a perfect matching that contains a matching cut. Both Matching Cut and Disconnected Perfect Matching are NP-complete for planar graphs of girth 5, whereas Perfect Matching Cut is known to be NP-complete even for subcubic bipartite graphs of arbitrarily large fixed girth. We prove that Matching Cut and Disconnected Perfect Matching are also NP-complete for bipartite graphs of arbitrarily large fixed girth and bounded maximum degree. Our result for Matching Cut resolves a 20-year old open problem. We also show that the more general problem d-Cut, for every fixed $d \geq 1$, is NP-complete for bipartite graphs of arbitrarily large fixed girth and bounded maximum degree. Furthermore, we show that Matching Cut, Perfect Matching Cut and Disconnected Perfect Matching are NP-complete for H-free graphs whenever H contains a connected component with two vertices of degree at least 3. Afterwards, we update the state-of-the-art summaries for H-free graphs and compare them with each other, and with a known and full classification of the Maximum Matching Cut problem, which is to determine a largest matching cut of a graph G. Finally, by combining existing results, we obtain a complete complexity classification of Perfect Matching Cut for $\mathcal{H}$-subgraph-free graphs where $\mathcal{H}$ is any finite set of graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases matching cut, perfect matching, girth, H-free graph

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.31

Acknowledgements We thank Hoang-Oanh Le for a significant simplification of our original proof of Theorem 6, which we simplified a bit further. We thank Van Bang Le for observing the bound on the maximum degree in Theorem 6, solving an Open Problem Garden question.

1 Introduction

We consider classic graph problems for finding certain edge cuts, which have in common that their edges must form a matching. In order to explain this, let $G = (V, E)$ be a connected graph. A set $M \subseteq E$ is a *matching* of G if no two edges in M share an end-vertex; M is *perfect* if every vertex of G is incident to an edge of M. A set $M \subseteq E$ is an *edge cut* of G if V can be partitioned into two sets B and R, such that M consists of all the edges with one end-vertex in B and the other one in R. We say that M is a *(perfect) matching cut* of G if M is a (perfect) matching that is also an edge cut. We refer to Figure 1 for some examples.

Figure 1 The graph P_6 from [31] 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.

Graphs with matching cuts were introduced in 1970 by Graham [18] as *decomposable* graphs. Matching cuts have applications in number theory [18], graph drawing [34], graph homomorphisms [16], edge labelings [2] and ILFI networks [13]. Moreover, a connected graph with no vertex of degree 1 has a matching cut if and only if its line graph has a vertex cut that is an independent set (*stable cut set*). As such, (perfect) matching cuts are well studied in the literature.

Instead of considering perfect matchings that *are* edge cuts, we can also consider perfect matchings in graphs that *contain* edge cuts. Such perfect matchings are called *disconnected perfect matchings*; see Figure 1 again. Note that every perfect matching cut is a disconnected perfect matching. However, there exist connected graphs, like the cycle C_6 on six vertices, that have a disconnected perfect matching (and thus a matching cut) but no perfect matching cut. There also exist connected graphs, like the path P_3 on three vertices, that have a matching cut, but no disconnected perfect matching (and thus no perfect matching cut either).

The problems Matching Cut, Disconnected Perfect Matching and Perfect Matching are to decide if a connected graph has a matching cut, disconnected perfect matching or perfect matching cut, respectively. As explained below, all three problems are NP-complete, and have been extensively studied for special graph classes.

Our Focus. The *girth* of a graph that is not a forest is the number of edges of a shortest cycle in it; a forest has infinite girth. In 2003, Bonsma [6] asked if Matching Cut is still NP-complete for graphs of large girth and proved that this is indeed the case for planar graphs of girth 5. In the 2009 journal version of [6], Bonsma showed that every connected planar graph of girth at least 6 has a matching cut. Hence, the complexity status of Matching Cut for graphs of large girth remained unknown and was regularly posed as an open problem [9, 24, 27, 29].

Bouquet and Picouleau [8] proved that Disconnected Perfect Matching is NP-complete for planar graphs of girth $g = 5$ and left the cases where $g \geq 6$ open. In contrast, Le and Telle [27] proved that for every $g \geq 3$, Perfect Matching Cut is NP-complete for subcubic bipartite graphs of girth at least g (a graph is *subcubic* if it has maximum degree at most 3). We focus on the two remaining open problems:

What is the complexity of Matching Cut *and* Disconnected Perfect Matching *for graphs of large girth?*

A challenging task is to find gadgets with no edges that are subdivided twice; such gadgets always have a matching cut (take the two subdivision vertices on one side and all other vertices on the other) and cannot be used in any hardness reduction.

1.1 Other Relevant Known Results

We restrict ourselves to *hereditary* graph classes, that is, classes of graphs closed under vertex deletion. A class of graphs $\mathcal{G}$ is hereditary if and only if the graphs in $\mathcal{G}$ are $\mathcal{F}_\mathcal{G}$*-free* for some unique set $\mathcal{F}_\mathcal{G}$, that is, they do not contain any graph from $\mathcal{F}_\mathcal{G}$ as an induced subgraph. For a *systematic* study, one may start with H*-free* graphs (so where $\mathcal{F}_\mathcal{G}$ has a single graph H).

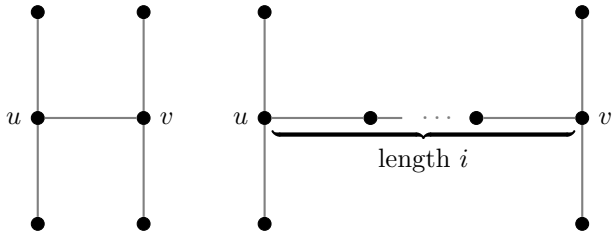


Figure 2 The graphs $H^* = H_1^*$ (left) and H_i^* (right).

Matching Cuts. Chvátal [10] proved that Matching Cut is NP-complete even for $K_{1,4}$-free graphs of maximum degree 4 (the graph $K_{1,r}$ is the $(r+1)$-vertex star); see [34] for an alternative hardness proof. In contrast, Chvátal [10] also proved that Matching Cut is polynomial-time solvable for graphs of maximum degree at most 3, whereas Bonsma [6] proved the same for $K_{1,3}$-free graphs, thereby generalizing a known result of Moshi [32] for line graphs, and also for P_4-free graphs; the latter result was extended to P_5-free graphs in [14] and to P_6-free graphs in [29]. Kratsch and Le [23] proved polynomial-time solvability for $(K_{1,4}, K_{1,4}+e)$-free graphs. It is also known 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 [29]; for two vertex-disjoint graphs G_1 and G_2, we write $G_1+G_2 = (V(G_1) \cup V(G_2), E(G_1) \cup E(G_2))$.

Moshi [32] proved that Matching Cut is NP-complete even for bipartite graphs where the vertices in one set of the bipartition all have degree exactly 2. Consequently, Matching Cut is NP-complete for $(H_1^*, H_3^*, H_5^*, \ldots)$-free bipartite graphs, where $H_1^* = H^*$ denotes the graph that looks like the letter "H", and for $i \geq 2$, the graph H_i^* is the graph obtained from $H^* = H_1^*$ by subdividing the middle edge of H_1^* exactly $i-1$ times; see also Figure 2.

Le and Randerath [26] proved that Matching Cut is NP-complete for $K_{1,5}$-free bipartite graphs and thus for C_s-free graphs if s is odd. Recall that Bonsma [6] proved that Matching Cut is NP-complete for planar graphs of girth 5, and thus for (C_3, C_4)-free graphs. By using the graph transformation of Moshi [32], Matching Cut is NP-complete for C_s-free graphs also if s is even and at least 6 [29]. In [31] it was shown that Matching Cut is NP-complete even for $(3P_5, P_{15})$-free graphs, strengthening a result of [14]. Afterwards, Le and Le [25] proved that Matching Cut is NP-complete even for $(3P_6, 2P_7, P_{14})$-free graphs.

We refer to [7, 24, 29] for results for non-hereditary graph classes, to [3, 15, 17, 22, 23] for parameterized complexity results and exact algorithms, to [4, 17] for a generalization of Matching Cut to d-Cut (where instead of at most one neighbour we allow each vertex to have at most d neighbours across the cut) and to [9] for a comprehensive overview.

Disconnected Perfect Matchings. The Disconnected Perfect Matching problem was introduced by Bouquet and Picouleau [8]. They used a different name but we adapt the name of Le and Telle [27] to avoid confusion with Perfect Matching Cut. Bouquet and Picouleau [8] showed that Disconnected Perfect Matching is, among others, polynomial-time solvable for $K_{1,3}$-free graphs and P_5-free graphs, but NP-complete for $K_{1,4}$-free planar graphs, planar graphs of girth 5 and for bipartite graphs, and thus for C_s-free graphs for every odd s. Le and Le [25] proved that Disconnected Perfect Matching is NP-complete even for $(3P_6, 2P_7, P_{14})$-free graphs, which improved the previous NP-completeness result of [31] for $(3P_7, P_{19})$-free graphs.

Perfect Matching Cuts. The Perfect Matching Cut problem was first shown to be NP-complete by Heggernes and Telle [19]. Le and Telle [27] proved, besides NP-completeness for subcubic bipartite graphs of girth at least g (for any $g \geq 3$), that Perfect Matching

Cut is polynomial-time solvable for chordal graphs and for $S_{1,2,2}$-free graphs; the graph $S_{1,2,2}$ is obtained by subdividing two of the edges of the claw $K_{1,3}$ exactly once. In [31], it was shown that Perfect Matching Cut is polynomial-time solvable for P_6-free graphs, and moreover for $(H + P_4)$-free graphs if it is polynomial-time solvable for H-free graphs.

Even more recently, two new results were shown. Le and Le [25] proved that Perfect Matching Cut is NP-complete even for $(3P_6, 2P_7, P_{14})$-free graphs (using the same construction as for Matching Cut and Disconnected Perfect Matching), whereas Bonnet, Chakraborty and Duron [5] proved that Perfect Matching Cut is NP-complete for 3-connected cubic bipartite planar graphs.

1.2 Our Results

As our main results, we solve, in Section 3, the aforementioned two open problems in the literature [6, 8, 9, 24, 27, 29] by showing that for every $g \geq 3$, Matching Cut is NP-complete for bipartite graphs of girth at least g and maximum degree at most 60 and Disconnected Perfect Matching is NP-complete for bipartite graphs of girth at least g and maximum degree at most 74. Note that our first result answers, in the negative, a question [33] from the *Open Problem Garden*:

"For every d does there exists a g such that every graph with average degree smaller than d and girth at least g has a matching-cut?"

As an immediate consequence of our second result, we have that Disconnected Perfect Matching is NP-complete for C_s-free graphs for even s (as mentioned above, previously this was only known for odd s [8]). To overcome the obstacle that connected graphs with a 2-subdivided edge have a matching cut, we use results from the theory of expander graphs. The proof of our result on Matching Cut is surprisingly short. Moreover, we can extend it in a straightforward way to show the following. For every $d \geq 2$ and $g \geq 3$, there exists a function $f(d)$ that only depends on d, such that even d-Cut is NP-complete for bipartite graphs of girth at least g and maximum degree at most $f(d)$; recall that 1-Cut is the Matching Cut problem.

In Section 3, we also highlight an implicit result of Le and Telle [27] for Perfect Matching Cut. In their NP-hardness proof for subcubic bipartite graphs of high girth, they showed that a graph G has a *perfect* matching cut if and only if the graph obtained from G by subdividing some edge four times has a *perfect* matching cut. Since graphs with a 2-subdivided edge have a matching cut, this property shows a fundamental difference between matching cuts and perfect matching cuts.

For $i \geq 2$, recall from Figure 2 that H_i^* is the graph obtained from $H^* = H_1^*$ by subdividing the middle edge of H_1^* exactly $i-1$ times. Recall also that a result of Moshi [32] implies that Matching Cut is NP-complete for $(H_1^*, H_3^*, H_5^*, \ldots)$-free bipartite graphs. In Section 4, we extend this result by proving that for every $i \geq 1$, Matching Cut and Disconnected Perfect Matching are NP-complete for $(H_1^*, \ldots, H_i^*)$-free graphs. We obtain these results by replacing the gadget of Moshi [32] with more advanced graph transformations. In Section 4, we also make explicit that the construction of Le and Telle [27] for subcubic bipartite graphs of girth at least g gives in fact NP-completeness of Perfect Matching Cut for $(H_1^*, \ldots, H_i^*)$-free subcubic bipartite graphs of girth at least g, for any $g \geq 3$. Hence, all three problems are NP-complete for H-free graphs whenever H has a connected component with two vertices of degree at least 3.

In Section 5, we combine our new results with the known results. We update the state-of-the-art summaries for the three problems on H-free graphs; basically, for all three problems, we only need to consider cases where H is a linear forest. Apart from comparing the (partial)

classification of the three problems with each other, we also compare them with a recent complete classification of the Maximum Matching Cut problem for H-free graphs [30]. This problem is to determine a matching cut in a graph G with a maximum number of edges (or output that no matching cut exists in G). Finally, we show how the results for Perfect Matching Cut lead to a full classification of Perfect Matching Cut on $\mathcal{H}$-subgraph-free graphs, for every finite set of graphs $\mathcal{H}$.

2 Preliminaries

We only consider finite, undirected graphs without multiple edges and self-loops. Let $G = (V, E)$ be a graph. For $u \in V$, 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. For an integer $p \geq 0$, G is p-regular if every $u \in V$ has degree p. 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 $G - S = G[V \setminus S]$. 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 subdivision of an edge $e = uv$ of G replaces e by a new vertex w and edges uw and wv.

We will now define some useful colouring terminology for matching cuts used in other papers as well (see e.g. [31]). In the remainder of this section, we let $G = (V, E)$ be a connected graph. A *red-blue colouring* of G colours every vertex of G either red or blue. If every vertex of some set $S \subseteq V$ has the same colour (red or blue), then S is said to be *monochromatic*. We also say that $G[S]$ is *monochromatic*. A red-blue colouring of G is *valid* if the following holds:

1. every blue vertex has at most one red neighbour;
2. every red vertex has at most one blue neighbour; and
3. both colours red and blue are used at least once.

If a red vertex u in G has a blue neighbour v, then u and v are said to be *matched*. See Figure 1 for three examples of valid red-blue colourings of the P_6.

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. Note that $V = R \cup B$. 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 $B' = B$; see Figure 1 (middle) for an example of a perfect red-blue colouring (of the P_6). A red-blue colouring of G is *perfect-extendable* if it is valid and $G[R \setminus R']$ and $G[B \setminus B']$ both contain a perfect matching; see Figure 1 (right) for an example of a perfect-extendable red-blue colouring (of the P_6). In other words, the matching defined by the edges with one end-vertex in R' and the other one in B' can be extended to a perfect matching in G or, equivalently, is contained in a perfect matching in G.

We now make the following straightforward observation (see also e.g. [31]).

▶ **Observation 1.** *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.*

Finally, we formally define the notion of a d-cut. For an integer $d \geq 1$ and a connected graph $G = (V, E)$, a set $M \subseteq E$ is a *d-cut* of G if V can be partitioned into two sets B and R, such that the following two conditions hold:

(i) M consists of all the edges with one end-vertex in B and the other one in R; and

(ii) every vertex in B has at most d neighbours in R, and vice versa.

Recall that the corresponding d-Cut problem is to decide if a connected graph has a d-cut. Hence, 1-Cut and Matching Cut are the same problems.

3 Hardness for Arbitrary Given Girth

In this section, we will show that Matching Cut (Section 3.1) and Disconnected Perfect Matching (Section 3.2) remain NP-complete even for graphs of high girth and bounded maximum degree; recall that previously both problems were known to be NP-complete for (planar) graphs of girth at least g, only for $g \leq 5$ [6, 8]. We also briefly discuss how an implicit observation of Le and Telle [27] gives a simple, alternative proof for showing NP-completeness for Perfect Matching Cut (Section 3.3) for graphs of high girth.

We need the following notions. The *edge expansion* $h(G)$ of a graph $G = (V, E)$ on n vertices is defined as

$$h(G) = \min_{1 \leq |S| \leq \frac{n}{2}} \frac{|\partial S|}{|S|},$$

where $\partial S := \{uv \in E \mid u \in S, v \in V \setminus S\}$ is the set of all edges of G with one end-vertex in S and one end-vertex outside S.

A connected graph G is said to be *(matching-)immune* if G admits no matching cut. We generalize this notion as follows. For an integer $d \geq 1$, we say that a connected graph G is *d-immune* if G admits no d-cut, so being 1-immune is the same as being immune. We make the following observation (proof omitted).

▶ **Observation 2.** *For every integer $d \geq 1$, every connected graph G with $h(G) > d$ is d-immune.*

3.1 Matching Cut and d-Cut

In order to prove our hardness results for Matching Cut and d-Cut for bipartite graphs of high girth and bounded maximum degree, we use known results on expander graphs and number theory.

Two integers a and b are *coprime*, if they do not have a common divisor greater than 1. The following result is well known.

▶ **Theorem 3** ([11]). *For two positive, coprime integers a and b, the sequence $a + bk$, for $k \in \mathbb{N}$ contains infinitely many primes.*

For an integer a and a prime p, the *Legendre symbol* is defined as $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \bmod p$. It is well known (see, for example, [35]) that for any integer a and prime p, it holds that $\left(\frac{a}{p}\right) \equiv 0 \bmod p$, or $1 \bmod p$, or $(p-1) \bmod p$, and with slight abuse of notation one denotes these integers by 0, 1 and -1, respectively.

We use Theorem 3 to show the following lemma (proof omitted).

▶ **Lemma 4.** *There are infinitely many primes q, such that*

a) $q > 13$,

b) $q \equiv 1 \bmod 4$, *and*

c) *the Legendre symbol* $\left(\frac{q}{13}\right) = -1$.

Figure 3 The graph F with designated vertices x and y at distance at least g (left) and the graph F' (right) from the proof of Theorem 6.

Lemma 5 uses known results from the theory of expander graphs. We need the fact that the graph in the statement of Lemma 5 has a perfect matching only in Section 3.2.

▶ **Lemma 5.** *For every $g \geq 3$, there is an immune 14-regular bipartite graph with girth at least g that contains a perfect matching.*

Proof. It follows from a construction of Lubotzky, Phillips and Sarnak [28] based on Caley graphs that for every two primes p and q with the following four properties

- $q > p$,
- $p \equiv 1 \bmod 4$,
- $q \equiv 1 \bmod 4$, and
- $\left(\frac{p}{q}\right) = -1$,

there exists a $(p+1)$-regular bipartite graph G with $\lambda_2(G) \leq 2\sqrt{p}$ and girth at least $4\log_p q - \log_p 4$; here, $\lambda_2(G)$ denotes the second largest eigenvalue of the adjacency matrix of G.

We set $p = 13$, so $p \equiv 1 \bmod 4$. We now combine the above with Lemma 4 to find that there exist infinitely many primes q, such that there exists a 14-regular bipartite graph G_q with $\lambda_2(G_q) \leq 2\sqrt{13}$ and girth at least $4\log_{13} q - \log_{13} 4$. Moreover, Dodziuk [12] and, independently, Alon and Milman [1] showed that for every integer $\ell \geq 1$, every ℓ-regular graph G satisfies $h(G) \geq \frac{1}{2}(\ell - \lambda_2(G))$. This means that

$$h(G_q) \geq \frac{1}{2}(14 - \lambda_2(G_q)) \geq \frac{1}{2}(14 - 2\sqrt{13}) \approx 3.39 > 1.$$

Hence, by applying Observation 2, we find that G_q is immune.

By taking q sufficiently large, we conclude that for any $g \geq 3$, there exists an immune 14-regular bipartite graph G with girth at least g. Finally, as G is bipartite and regular, we find that G has a perfect matching (due to Hall's Marriage Theorem). ◀

▶ Remark 1. The arguments that we used to prove Lemma 5 do not allow us to set $p = 13$ to a smaller value. We need p to be prime, and moreover, it must hold that $p \equiv 1 \bmod 4$. Hence, the only alternative value for p that is smaller than 13 would be $p = 5$. However, $p = 5$ yields $h(G_q) \geq \frac{1}{2}(6 - 2\sqrt{5}) \approx 0.76$, so we cannot conclude from this that G_q is immune.

We use Lemma 5 in the proof of our first main result.

▶ **Theorem 6.** *For every integer $g \geq 3$,* Matching Cut *is* NP*-complete for bipartite graphs of girth at least g and maximum degree at most* 60*.*

Proof. Let $g \geq 3$. As the class of graphs of girth at least $g+1$ is a subclass of the class of graphs of girth at least g, we may assume without loss of generality that g is divisible by 2 but not by 4, so $\frac{g}{2}$ is odd. We reduce from Matching Cut. Recall that Matching Cut is

NP-complete for graphs of degree at most 4 [10]. Let G be a connected graph of maximum degree at most 4. From G we construct a graph G' with the required properties, but first we define an auxiliary graph F'.

By Lemma 5 there exists an immune 14-regular bipartite graph F with girth at least g. Note that F has constant size, so we can find F in constant time.

Let x and y be two vertices of distance $\frac{g}{2}$ in F. As $\frac{g}{2}$ is odd, x and y belong to opposite bipartition classes of F. We take two copies F_1 and F_2 of F, and we add an edge between the two copies x_1 and x_2 of x and an edge between the two copies y_1 and y_2 of y. This yields a graph F'. As F is bipartite and has girth at least g, we find that F' is bipartite and has girth at least g as well. The construction also gives us that x_1 and y_2 belong to the same bipartition class of F'. See also Figure 6.

Now consider an edge uv in G. The *F'-replacement* of uv is obtained from the graphs G and F' by identifying u with x_1 and v with y_2. We do an F'-replacement on every edge of G. This yields the graph G'. Since F' is bipartite such that the distance between x_1 and y_2 is even, we find that G' is bipartite as well. By construction, G' has girth at least g and maximum degree at most $4 \times (14 + 1) = 60$.

We claim that G has a matching cut if and only if G' has a matching cut. We prove this below, using Observation 1-(i) implicitly.

First suppose that G has a matching cut, so G has a valid red-blue colouring c. For every edge uv in G we do as follows. Let F' with copies F_1 and F_2 of F be the corresponding F'-replacement applied on uv. If u and v have the same colour, then we colour every vertex of $V(F') \setminus \{u, v\}$ with that colour. If u and v are coloured differently, say u is red and v is blue, then we colour every vertex in $V(F_1)$ red and every vertex in $V(F_2)$ blue. This yields a valid red-blue colouring of G'. Hence, G' has a matching cut.

Now suppose that G' has a matching cut, so G' has a valid red-blue colouring c'. As F is immune, every copy of it in G' is monochromatic. Thus, for two vertices $x_1, y_2 \in V(F')$ in some graph F' in G', it holds that x_1 and y_2 each have a neighbour in F' of the opposite colour if and only if x_1 and y_2 have different colours in G'. Hence, the restriction of c' to $V(G)$ is a valid red-blue colouring of G. Hence, G has a matching cut. ◀

We now focus on the d-Cut problem for arbitrary $d \geq 1$, and we show how Theorem 6 can be generalized in a straightforward way. This requires us to replace Lemma 4 by the following lemma (proof omitted).

▶ **Lemma 7.** *There are infinitely many primes p, such that*

a) $p \geq 13$,

b) $p \equiv 1 \bmod 4$, *and*

c) *there exists an infinite set Q_p such that the following holds for every $q \in Q_p$:*

- **i.** $q > p$
- **ii.** $q \equiv 1 \bmod 4$, *and*
- **iii.** *the Legendre symbol* $\left(\frac{q}{p}\right) = -1$.

We are now ready to generalize Theorem 6.

▶ **Theorem 8.** *For every integer $d \geq 1$ and every integer $g \geq 3$, there is a function $f(d)$ that only depends on d, such that d-Cut is* NP*-complete for bipartite graphs of girth at least g and maximum degree at most $f(d)$.*

Proof. Let $d \geq 1$. By Observation 2, every graph G with $h(G) > d$ is d-immune. Hence, we can construct a $(p+1)$-regular bipartite gadget F using the arguments from the proof of Lemma 5, where Lemma 7 plays the role of Lemma 4. That is, we can choose p such that

(i) p is a prime congruent to 1 mod 4, and

(ii) $\frac{1}{2}(p + 1 - \lambda_2(G)) \geq \frac{1}{2}(p + 1 - 2\sqrt{p}) > d$.

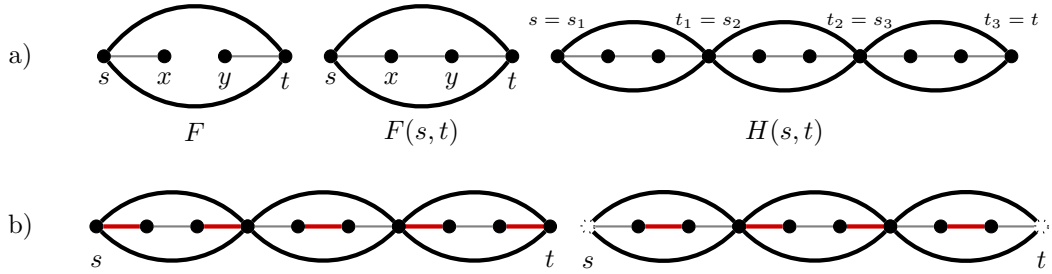


Figure 4 a) Illustration of the graphs F (left), $F(s,t)$ (middle) and $H(s,t)$ (right). b) Illustration of how the edges in the perfect matching of $H(s,t)$ (left) resp. $H(s,t) - \{s,t\}$ (right) are chosen (these edges are represented as red, thick edges).

We now reduce from d-Cut. Gomes and Sau [17] proved that d-Cut is NP-hard for graphs of maximum degree at most $2d+2$. Hence, we may assume that the instance G from d-Cut has maximum degree at most $2d+2$. By applying the arguments of the proof of Theorem 6, we can use F and G to construct for every integer $g \geq 3$, a bipartite graph G' of girth at least g and maximum degree at most $f(d)$, such that G has a d-cut if and only if G' has a d-cut. ◀

▶ Remark 2. Since it is not possible to give the smallest prime p satisfying conditions (i) and (ii) in the proof of Theorem 8, we can merely state that the maximum degree of G' is bounded by some function $f(d)$ that only depends on d.

3.2 Disconnected Perfect Matching

We now show that Disconnected Perfect Matching is NP-complete for graphs of arbitrarily large fixed girth and bounded maximum degree. Our proof uses some new ideas, but we note that it is also possible to use a similar approach as in the proof of Theorem 6. However, this will lead to a slightly worse bound on the maximum degree, namely 75 instead of 74.

We first define some useful auxiliary graphs. Fix $g \geq 3$ such that g is divisible by 6 and $g/6$ is odd. Let F be a 14-regular immune bipartite graph with girth at least $2(g+1)$ containing a perfect matching. We note that F exists by Lemma 5, and as F has constant size, F can be found in constant time.

Let s and t be two designated vertices in F at distance at least $g+1$. We fix a perfect matching M of F. Let $x \in N_F(s)$ and $y \in N_F(t)$ be the (unique) neighbours of s and t in M. Then, since s and t are at distance at least $g+1$, x and y are at distance at least $g-1$. We add the edge xy and denote the resulting graph by $F(s,t)$; see also Figure 4a).

We make the following observation (proof omitted).

▶ **Lemma 9.** *The graph $F(s,t)$ is immune, bipartite and has girth at least g. Moreover, both $F(s,t)$ and $F(s,t) - \{s,t\}$ contain a perfect matching.*

We now take $k = \frac{g}{6}$ copies $F(s_1,t_1), \ldots, F(s_k,t_k)$ of $F(s,t)$ and identify s_{i+1} with t_i for all $i \in \{1,\ldots,k-1\}$. We set $s = s_1$ and $t = t_k$ and call the resulting graph $H(s,t)$; see also Figure 4a).

In the following lemma, whose proof we omit, we show some useful properties of $H(s,t)$ and $H(s,t) - \{s,t\}$.

▶ **Lemma 10.** *The graph $H(s,t)$ is immune, bipartite, and has girth at least g. Moreover, $\text{dist}(s,t) \geq \frac{g}{2}$, and both $H(s,t)$ and $H(s,t) - \{s,t\}$ contain a perfect matching.*

We are now ready to prove the following result.

▶ **Theorem 11.** *For every integer $g \geq 3$,* Disconnected Perfect Matching *is* NP*-complete for bipartite graphs of girth at least g and maximum degree at most* 74*.*

Proof. Let $g \geq 3$. We reduce from Matching Cut for bipartite graphs of girth at least g and maximum degree at most 60, which is NP-complete by Theorem 6. Similar to Theorem 6, we may assume without loss of generality that g is divisible by 12. Let G be a bipartite graph of girth at least g and maximum degree 60. We construct a graph G' by taking two copies G_1 and G_2 of G, where we connect every vertex $v \in V(G_1)$ and its copy $v' \in V(G_2)$ using the graph $H(v,v')$.

To see that G' has girth at least g, we consider first G_1 and G_2, which both have girth at least g. Any cycle containing vertices from both copies has to pass twice through a graph $H(s,t)$. Thus, it will always have length at least g, and so G' has girth g. Every vertex inside $H(s,t)$ has degree at most 28, whereas s and t only have degree 14. The degree of a vertex $v \in V(G_1) \cup V(G_2)$ is the degree of the vertex in the original graph G plus the degree in the graph $H(v,v')$. Thus, the degree of v is at most 74.

We also claim that G' is bipartite. For a contradiction, assume that G' has an odd cycle C. As G_1 and G_2 are both bipartite, C' must contain vertices from G_1 and G_2. Note that C passes through an even number of graphs $H(v,v')$, where $v \in V(G_1)$ and $v' \in V(G_2)$. Hence, the number of edges in $E(G) \setminus (E(G_1) \cup E(G_2))$ is even. Since the graph $H(v,v')$ always connects a vertex v in G_1 and its copy v' in G_2 we can find an odd cycle C' in G_1, consisting of the edges in $C \cap E(G_1)$ and the edges in $E(G_1)$ corresponding to the edges from $C \cap E(G_2)$, a contradiction.

Finally, we show that G admits a matching cut if and only if G' admits a disconnected perfect matching. Consider some vertex $v \in V(G_1)$ and its copy $v' \in V(G_2)$. Since v and v' are connected by the graph $H(v,v')$, which is immune, v and v' will always have the same colour in any valid red-blue colouring of G'. Thus, G_1 and G_2 will be coloured the same in any valid red-blue colouring. Now, if G admits a perfect-extendable red-blue colouring, then G admits a valid red-blue colouring, as it suffices to colour G the same as G_1 (or G_2).

Conversely, if G admits a valid red-blue colouring, then we obtain a perfect-extendable colouring of G' as follows. We colour G_1 and G_2 the same as G. Notice that we colour the immune graphs connecting two copies of the same vertex such that they are monochromatic. This gives us a valid red-blue colouring of G', i.e. a matching cut M in G'. It remains to show that the matching cut is contained in a perfect matching of G'. Since the colourings of G_1 and G_2 are the same, we have that whenever a blue vertex $v \in V(G_1)$ is matched with a red vertex $u \in V(G_1)$, i.e. $vu \in M$, then their copies $v' \in V(G_2)$ and $u' \in V(G_2)$ are matched as well, i.e. $v'u' \in M$. By Lemma 10, we know that $H(v,v') - \{v,v'\}$ and $H(u,u') - \{u,u'\}$ contain both a perfect matching which we may add to M. For every $v \in V(G_1)$ with no neighbour of the other colour, we know that its copy $v' \in V(G_2)$ has no neighbour of the other colour either. Thus, we can use that $H(v,v')$ contains a perfect matching by Lemma 10 and add it to M. Repeatedly doing this yields a perfect matching of G' containing M. ◀

3.3 Perfect Matching Cut

In any perfect red-blue colouring of a connected graph $G = (V,E)$, a vertex $v \in V$ of degree 2 has exactly one neighbour coloured the same as itself and exactly one neighbour coloured differently than itself. One can use this observation to prove the following lemma. This lemma was implicit in [27] and we omit its proof.

▶ **Lemma 12** ([27]). *Let $G = (V', E')$ be the graph obtained from a connected graph G by 4-subdividing an edge e of G. Now, G has a perfect matching cut if and only if G' has a perfect matching cut.*

A simple proof for showing that Perfect Matching Cut is NP-complete for graphs of girth at least g is to apply Lemma 12, say, g times on each edge of the input graph. Recall that the more involved gadget of Le and Telle [27] yields NP-completeness even for subcubic bipartite graphs of girth at least g.

4 Hardness for Forbidden Subdivided H-Graphs

We show that Matching Cut (Section 4.1), Disconnected Perfect Matching (Section 4.2) and Perfect Matching Cut (Section 4.3) are NP-complete for $(H_1^*, \ldots, H_i^*)$-free graphs, for every $i \geq 1$.

4.1 Matching Cut

Let uv be an edge of a graph G. We define an edge operation as displayed in Figure 5, which when applied on uv will replace uv in G by the subgraph T_{uv}^i. Note that in the new graph, the only vertices from T_{uv}^i that may have neighbours outside T_{uv}^i are u and v.

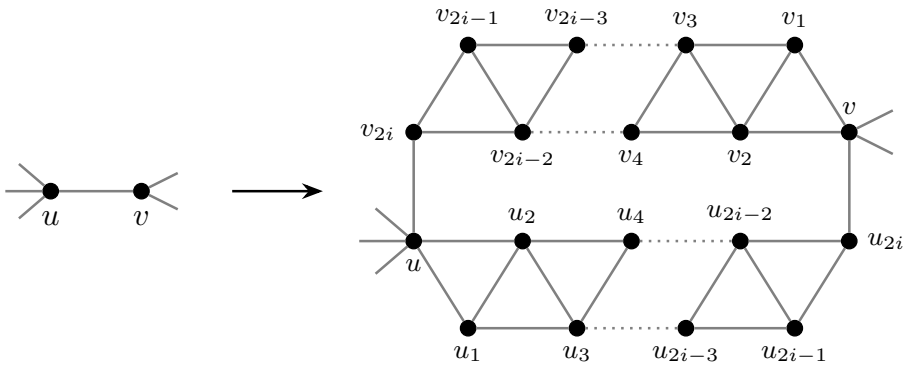


Figure 5 The edge uv (left) which we replace by the subgraph T_{uv}^i (right).

▶ **Theorem 13.** *For all $i \geq 1$, Matching Cut is NP-complete for $(H_1^*, \ldots, H_i^*)$-free graphs.*

Proof. Fix $i \geq 1$. Reduce from Matching Cut. Let $G = (V, E)$ be a connected graph. Replace every $uv \in E$ by the graph T_{uv}^i (see Figure 5). This yields the graph $G' = (V', E')$. We claim that G' is $(H_1^*, \ldots, H_i^*)$-free. For a contradiction, assume that G' contains an induced $H_{i'}^*$ for some $1 \leq i' \leq i$. Then G' contains two vertices x and y that are centers of an induced claw, as well as an induced path from x to y of length i'. All vertices in $V' \setminus V$ are not centers of any induced claw. Hence, x and y belong to V. By construction, any shortest path between two vertices of V has length at least $i + 1$ in G', a contradiction.

We claim that G' has a matching cut if and only if G has a matching cut. First suppose G' has a matching cut M', so G' has a valid red-blue colouring c'. We prove a claim for G':

▷ Claim 13.1. For every edge $uv \in E(G)$ it holds that

(a) either $c'(u) = c'(v)$, and then T_{uv}^i is monochromatic, or

(b) $c'(u) \neq c'(v)$, and then c' colours $u_1, \ldots, u_{2i}$ with the same colour as u, while c' colours all vertices of $T_{uv}^i - \{u, u_1, \ldots, u_{2i}\}$ with the same colour as v, and moreover, $uv_{2i}, vu_{2i} \in M'$.

Proof. First assume $c'(u) = c'(v)$, say c' colours u and v red. As any clique of size at least 3 is monochromatic, all vertices in T^i_{uv} are coloured red, so T^i_{uv} is monochromatic.

Now assume $c'(u) \neq c'(v)$, say u is red and v blue. As before, we find that all vertices $u_1, \ldots, u_{2i}$ have the same colour as u, so are red, while all vertices $v_1, \ldots, v_{2i}$ have the same colour as v, so are blue. By definition, every edge xy with $c'(x) \neq c'(y)$ belongs to M', so $uv_{2i}, vu_{2i} \in M'$. ◁

We construct a subset $M \subseteq E$ in G as follows. We add an edge $uv \in E$ to M if and only if $c'(u) \neq c'(v)$ in G'. We now show that M is a matching in G. Let $u \in V$. For a contradiction, suppose that M contains edges uv and uw for $v \neq w$. Then $c'(u) \neq c'(v)$ and $c'(u) \neq c'(w)$. By Claim 13.1, we find that M' matches u in G' to vertices in T^i_{uv} and T^i_{uw}, contradicting our assumption that M' is a matching (cut). Hence, M is a matching.

Now let c be the restriction of c' to V. If c colours every vertex of G with one colour, say red, then c' would also colour every vertex of G' red by Claim 13.1, contradicting the validity of c'. Hence, c uses both colours. Moreover, for every $uv \in E$, the following holds: if $c(u) \neq c(v)$, then $c'(u) \neq c'(v)$ and thus $uv \in M$. Hence, as M is a matching, c is valid, and thus M is a matching cut of G.

Conversely, assume that G admits a matching cut, so V has a valid red-blue colouring c. We construct a red-blue colouring c' of V' as follows.

- For every edge $uv \in E$ with $c(u) = c(v)$, we let $c'(x) = c(u)$ for every $x \in V(T^i_{uv})$.
- For every edge $uv \in E$ with $c(u) \neq c(v)$, we let $c'(u) = c'(u_1) = \cdots = c'(u_{2i}) = c(u)$ and $c'(v) = c'(v_1) = \cdots = c'(v_{2i}) = c(v)$.

As c is valid, c uses both colours and thus by construction, c' uses both colours. Let $u \in V$. Again as c is valid, $c(u) \neq c(v)$ holds for at most one neighbour v of u in G. Hence, by construction, u belongs to at most one non-monochromatic gadget T^i_{uv}. Thus, c' colours in G' at most one neighbour of u with a different colour than u. Let $u \in V' \setminus V$. By construction, we find again that c' colours at most one neighbour of u with a different colour than u. Hence, c' is valid, and so G' has a matching cut. ◀

4.2 Disconnected Perfect Matching

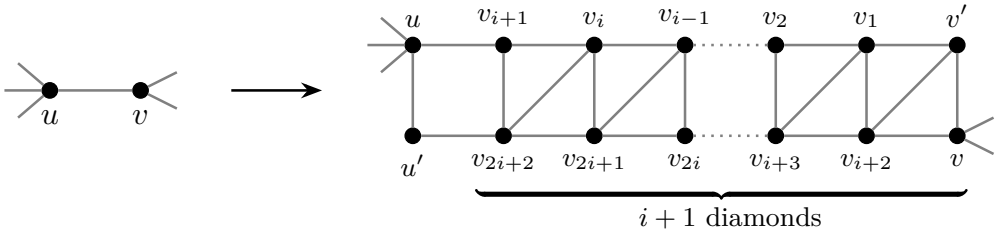


Figure 6 The edge uv (left) which we replace by the subgraph G^i_{uv} (right).

We can prove the following result in a similar way as Theorem 13 and give a sketch of its proof.

▶ **Theorem 14.** *For every $i \geq 1$,* Disconnected Perfect Matching *is* NP*-complete for $(H^*_1, \ldots, H^*_i)$-free graphs.*

Proof sketch. We first define a graph operation. Let uv be an edge of a graph G. We define an edge operation as displayed in Figure 6, which when applied on uv replaces uv by the subgraph G^i_{uv} for some integer $i \geq 1$. In the new graph, the only vertices from G^i_{uv} that may have neighbours outside G^i_{uv} are u and v.

Now fix an integer $i \geq 1$. As the class of $(H_1^*, \ldots, H_i^*)$-free graphs is contained in the class of $(H_1^*, \ldots, H_{i-1}^*)$-free graphs if $i \geq 2$, we may assume without loss of generality that i is even (we need this assumption at a later place in our proof). We reduce from Disconnected Perfect Matching itself. Let $G = (V, E)$ be a connected graph. We replace every edge $uv \in E$ by the graph G_{uv}^i (see Figure 6). Let $G' = (V', E')$ be the resulting graph.

We can show that G' is $(H_1^*, \ldots, H_i^*)$-free and that G has a disconnected perfect matching if and only if G' has a disconnected perfect matching. ◀

4.3 Perfect Matching Cut

We apply Lemma 12 (implicit in [27]) sufficiently times on every edge of a subcubic bipartite graph of girth at least g and combine this with the NP-completeness of Perfect Matching Cut for the class of such graphs [27]. Hence, Le and Telle essentially proved the following:

▶ **Theorem 15** ([27]). *For every $i \geq 1$ and $g \geq 3$, Perfect Matching Cut is NP-complete for $(H_1^*, \ldots, H_i^*)$-free subcubic bipartite graphs of girth at least g.*

5 Consequences and Open Problems

We give some consequences of our new results on Matching Cut, Disconnected Perfect Matching and Perfect Matching Cut for H-free graphs and H-subgraph-free graphs.

5.1 H-Free Graphs

We give three up-to-date classifications for H-free graphs by combining the results from [6, 8, 10, 25, 26, 31, 29, 27, 32] (see Section 1.1) with our new results. That is, we took the three state-of-the-art theorems in [31] and added both the result for H_i^*-free graphs and the result for $(3P_6, 2P_7, P_{14})$-free graphs from [25]. For Disconnected Perfect Matching we also added the new result for C_s-free graphs for even s, as implied by our girth result. We also compare the three partial classification with a recent, full classification of the optimization problem Maximum Matching Cut [30]. We write $G' \subseteq_i G$ if G' is an induced subgraph of G and $G' \supseteq_i G$ if G is an induced subgraph of G'.

▶ **Theorem 16.** *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 $H \supseteq_i K_{1,4}$, P_{14}, $3P_5$, $2P_7$, C_r for some $r \geq 3$ or H_j^* for some $j \geq 1$.*

▶ **Theorem 17.** *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 $H \supseteq_i K_{1,4}$, P_{14}, $3P_6$, $2P_7$, C_r for some $r \geq 3$ or H_j^* for some $j \geq 1$.*

▶ **Theorem 18.** *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 $H \supseteq_i K_{1,4}$, P_{14}, $3P_6$, $2P_7$, C_r for some $r \geq 3$ or H_j^* for some $j \geq 1$.*

▶ **Theorem 19.** *For a graph H, Maximum Matching Cut on H-free graphs is*
- *polynomial-time solvable if $H \subseteq_i sP_2 + P_6$ for some $s \geq 0$, and*
- *NP-hard if $H \supseteq_i K_{1,3}$, $2P_3$, C_r for some $r \geq 3$.*

A *subdivided claw* is a graph obtained from the claw $K_{1,3}$ by subdividing each of its edges zero or more times. Let $\mathcal{S}$ be the class of graphs, each connected component of which is either a path or a *subdivided claw*. From Theorem 16, it follows that Matching Cut is NP-complete for H-free graphs if H has a cycle, a vertex of degree at least 4, or a connected component

with two vertices of degree 3. Hence, the remaining open cases for Matching Cut on H-free graphs are all restricted to cases where H is a graph from $\mathcal{S}$. The same remark holds for Disconnected Perfect Matching due to Theorem 17, and for Perfect Matching Cut due to Theorem 18.

5.2 H-subgraph-free Graphs

For a graph H, a graph G is *H-subgraph-free* if G does not contain H as a subgraph. Every H-subgraph-free graph is H-free, whereas the reverse direction only holds if H is a complete graph. For a set $\mathcal{H}$ of graphs, a graph G is *$\mathcal{H}$-subgraph-free* if G is H-subgraph-free for every $H \in \mathcal{H}$.

For an integer p, a *p-subdivision* of an edge uv in a graph replaces uv by a path from u to v of length $p+1$. The *p-subdivision* of a graph G is obtained from G by p-subdividing each edge of G. For a graph class $\mathcal{G}$, we let $\mathcal{G}^p$ consist of all the p-subdivisions of the graphs in $\mathcal{G}$. A graph problem Π is NP-complete *under edge subdivision of subcubic graphs* if there is an integer $q \geq 1$ such that the following holds: if Π is NP-complete for the class $\mathcal{G}$ of subcubic graphs, then Π is NP-complete for $\mathcal{G}^{pq}$ for every $p \geq 1$. Now, Π is a *C123-problem* if (C1) Π is polynomial-time solvable for graphs of bounded treewidth; (C2) Π is NP-complete for subcubic graphs; and (C3) Π is NP-complete under edge subdivision of subcubic graphs.

In [20], it was shown that for every finite set of graphs $\mathcal{H}$, any C123-problem Π on $\mathcal{H}$-subgraph-free graphs is polynomial-time solvable if $\mathcal{H}$ contains a graph from $\mathcal{S}$ and NP-complete otherwise. Le and Telle [27] observed that Perfect Matching Cut satisfies C1 and proved C2 and C3 (see Lemma 12). Applying the above meta-theorem from [20] yields:

▶ **Theorem 20.** *For any finite set of graphs $\mathcal{H}$,* Perfect Matching Cut *on $\mathcal{H}$-subgraph-free graphs is polynomial-time solvable if $\mathcal{H}$ contains a graph from $\mathcal{S}$ and* NP*-complete otherwise.*

5.3 Open Problems

Apart from completing the classifications of Theorems 16–18, we also pose the following open problem.

▶ **Open Problem 1.** *Classify the computational complexity of* Matching Cut *and* Disconnected Perfect Matching *for $\mathcal{H}$-subgraph-free graphs.*

We note that classifications of Matching Cut and Disconnected Perfect Matching are unknown even for H-subgraph-free graphs (so when we forbid only a single graph H as a subgraph). So far, only a partial classification for Matching Cut restricted to $\mathcal{H}$-subgraph-free graphs has been shown [21]. In particular, the transformations for Matching Cut and Disconnected Perfect Matching from Section 4 do not decrease the girth and yield graphs with many cycles of varying length as subgraphs. Hence, new techniques are needed.

Finally, with our current technique (Lemma 5) we cannot obtain a better bound on the maximum degree of the graphs of arbitrarily large fixed girth in the proofs of Theorems 6 and 11.

▶ **Open Problem 2.** *Can the two maximum degree bounds in Theorems 6 and 11 be improved?*

References

1 Noga Alon and Vitali D Milman. $\lambda 1$, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38:73–88, 1985.

2 Júlio Araújo, Nathann Cohen, Frédéric Giroire, and Frédéric Havet. Good edge-labelling of graphs. *Discrete Applied Mathematics*, 160:2502–2513, 2012.

3 N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare. Vertex partitioning problems on graphs with bounded tree width. *Discrete Applied Mathematics*, 319:254–270, 2022.

4 N. R. Aravind and Roopam Saxena. An FPT algorithm for Matching Cut and d-Cut. *Proc. IWOCA 2021, LNCS*, 12757:531–543, 2021.

5 Edouard Bonnet, Dibyayan Chakraborty, and Julien Duron. Cutting barnette graphs perfectly is hard. *Proc. WG 2023, LNCS*, to appear.

6 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 (conference version: WG 2003).

7 Mieczyslaw Borowiecki and Katarzyna Jesse-Józefczyk. Matching cutsets in graphs of diameter 2. *Theoretical Computer Science*, 407:574–582, 2008.

8 Valentin Bouquet and Christophe Picouleau. The complexity of the Perfect Matching-Cut problem. *CoRR*, abs/2011.03318, 2020. `arXiv:2011.03318`.

9 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.

10 Vasek Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8:51–53, 1984.

11 Peter G. Lejeune Dirichlet. Beweis des Satzes, dass jede unbegrenzte arithmetische Progression, deren erstes Glied und Differenz ganze Zahlen ohne gemeinschaftlichen Faktor sind, unendlich viele Primzahlen enthält. *Abhandlungen der Königlichen Preußischen Akademie der Wissenschaften zu Berlin*, 1837.

12 Jozef Dodziuk. Difference equations, isoperimetric inequality and transience of certain random walks. *Transactions of the American Mathematical Society*, 284:787–794, 1984.

13 Arthur M. Farley and Andrzej Proskurowski. Networks immune to isolated line failures. *Networks*, 12:393–403, 1982.

14 Carl Feghali. A note on Matching-Cut in P_t-free graphs. *Information Processing Letters*, 179:106294, 2023.

15 Petr A. Golovach, Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Refined notions of parameterized enumeration kernels with applications to matching cut enumeration. *Journal of Computer and System Sciences*, 123:76–102, 2022.

16 Petr A. Golovach, Daniël Paulusma, and Jian Song. Computing vertex-surjective homomorphisms to partially reflexive trees. *Theoretical Computer Science*, 457:86–100, 2012.

17 Guilherme Gomes and Ignasi Sau. Finding cuts of bounded degree: complexity, FPT and exact algorithms, and kernelization. *Algorithmica*, 83:1677–1706, 2021.

18 Ronald L. Graham. On primitive graphs and optimal vertex assignments. *Annals of the New York Academy of Sciences*, 175:170–186, 1970.

19 Pinar Heggernes and Jan Arne Telle. Partitioning graphs into generalized dominating sets. *Nordic Journal of Computing*, 5:128–142, 1998.

20 Matthew Johnson, Barnaby Martin, Jelle J. Oostveen, Sukanya Pandey, Daniël Paulusma, Siani Smith, and Erik Jan van Leeuwen. Complexity framework for forbidden subgraphs I: The framework. *CoRR*, abs/2211.12887, 2022. `arXiv:2211.12887`.

21 Matthew Johnson, Barnaby Martin, Sukanya Pandey, Daniël Paulusma, Siani Smith, and Erik Jan van Leeuwen. Complexity framework for forbidden subgraphs III: When problems are tractable on subcubic graphs. *Proc. MFCS 2023, LIPIcs*, 272:57:1–57:15, 2023.

22 Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Matching Cut: Kernelization, single-exponential time FPT, and exact exponential algorithms. *Discrete Applied Mathematics*, 283:44–58, 2020.

23 Dieter Kratsch and Van Bang Le. Algorithms solving the Matching Cut problem. *Theoretical Computer Science*, 609:328–335, 2016.

24 Hoang-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.

25 Hoàng-Oanh Le and Van Bang Le. Complexity results for matching cut problems in graphs without long induced paths. *Proc. WG 2023, LNCS*, to appear.

26 Van Bang Le and Bert Randerath. On stable cutsets in line graphs. *Theoretical Computer Science*, 301:463–475, 2003.

27 Van Bang Le and Jan Arne Telle. The Perfect Matching Cut problem revisited. *Proc. WG 2021, LNCS*, 12911:182–194, 2021.

28 Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.

29 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*, 936, 2022.

30 Felicia Lucke, Daniël Paulusma, and Bernard Ries. Dichotomies for Maximum Matching Cut: H-Freeness, Bounded Diameter, Bounded Radius. *Proc. MFCS 2023, LIPIcs*, 272:64:1–64:15, 2023.

31 Felicia Lucke, Daniël Paulusma, and Bernard Ries. Finding matching cuts in H-free graphs. *Algorithmica*, to appear.

32 Augustine M. Moshi. Matching cutsets in graphs. *Journal of Graph Theory*, 13:527–536, 1989.

33 Open problem garden. `http://www.openproblemgarden.org/op/matching_cut_and_girth`. (accessed on 22 June 2023).

34 Maurizio Patrignani and Maurizio Pizzonia. The complexity of the Matching-Cut problem. *Proc. WG 2001, LNCS*, 2204:284–295, 2001.

35 Harold N. Shapiro. Introduction to the Theory of Numbers. *Dover Publications*, 2008.

Computing Paths of Large Rank in Planar Frameworks Deterministically

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway

Petr A. Golovach
Department of Informatics, University of Bergen, Norway

Tuukka Korhonen
Department of Informatics, University of Bergen, Norway

Giannos Stamoulis
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Abstract

A framework consists of an undirected graph G and a matroid M whose elements correspond to the vertices of G. Recently, Fomin et al. [SODA 2023] and Eiben et al. [ArXiV 2023] developed parameterized algorithms for computing paths of rank k in frameworks. More precisely, for vertices s and t of G, and an integer k, they gave FPT algorithms parameterized by k deciding whether there is an (s,t)-path in G whose vertex set contains a subset of elements of M of rank k. These algorithms are based on Schwartz-Zippel lemma for polynomial identity testing and thus are randomized, and therefore the existence of a deterministic FPT algorithm for this problem remains open.

We present the first deterministic FPT algorithm that solves the problem in frameworks whose underlying graph G is planar. While the running time of our algorithm is worse than the running times of the recent randomized algorithms, our algorithm works on more general classes of matroids. In particular, this is the first FPT algorithm for the case when matroid M is represented over rationals.

Our main technical contribution is the nontrivial adaptation of the classic irrelevant vertex technique to frameworks to reduce the given instance to one of bounded treewidth. This allows us to employ the toolbox of representative sets to design a dynamic programming procedure solving the problem efficiently on instances of bounded treewidth.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Planar graph, longest path, linear matroid, irrelevant vertex

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.32

Related Version *Full Version*: `https://arxiv.org/abs/2305.01993`

Funding The research leading to these results has received funding from the Research Council of Norway via the project BWCA (grant no. 314528). Giannos Stamoulis acknowledges support by the ANR project ESIGMA (ANR-17-CE23-0010) and the French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027).

1 Introduction

A *framework* is a pair (G,M), where G is a graph and $M=(V(G),\mathcal{I})$ is a matroid on the vertex set of G. This term appears in the recent monograph of Lovász [39], where he defines frameworks as graphs with a collection of vectors of $\mathbb{R}^d$ labeling their vertices. Frameworks have appeared in the literature under many different names. For example, they are mentioned as *pregeometric graphs* in the influential work of Lovász [37] on representative families of linear matroids and as *matroid graphs* in the book of Lovász and Plummer [38].

The problem of computing maximum matching in frameworks is closely related to the matchoid, the matroid parity, and polymatroid matching problems (see [38] for an overview). More broadly, the problems of finding specific subgraphs of large ranks in frameworks belong to the wide family of problems about submodular function optimization under combinatorial constraints [7, 8, 14, 41].

Fomin et al. in [15] introduced the following Maximum Rank (s,t)-Path problem. In this problem, given a framework (G, M), two vertices s and t of G, and an integer k, we seek for an (s,t)-path in G where the rank function of M evaluates to at least k. We say that such a path *has rank at least* k.

Max Rank (s,t)-Path

Input:	A framework (G, M), vertices s and t of G, and an integer $k \geq 0$.
Task:	Decide whether G contains an (s,t)-path of rank at least k.

Max Rank (s,t)-Path encompasses several fundamental and well-studied problems about paths and cycles in undirected graphs.

Longest path. Of course, when M is a uniform matroid, then a path is of rank at least k if and only if it contains at least k vertices. In this case, we have the classical Longest Path problem, where for a graph G and integer k the task is to identify whether G contains a path with at least k vertices [1].

T-cycle. In this problem, we are given a set T of terminals and the task is to decide whether there is a cycle through all terminals [5, 25, 45]. T-cycle is the special case of Max Rank (s,t)-Path. Consider the following linear matroid. For every vertex of G not in T we assign a $|T|$-dimensional vector whose all entries are zero. To vertices of T we assign vectors forming an orthonormal basis of $\mathbb{R}^{|T|}$. Then G has a cycle passing through all terminals if and only if (G, M) has an (s,t)-path of rank $|T|$, for some $\{s,t\} \in E(G)$.

Maximum Colored Path. In the Maximum Colored (s,t)-Path problem, we are given a colored graph G, two vertices s and t of G, and an integer k. The task is to decide whether G has an (s,t)-path containing at least k different colors [6, 15] (see also [9, 10]). Maximum Colored (s,t)-Path is the special case of Max Rank (s,t)-Path where the matroid M is a partition matroid. Indeed, in this matroid the ground set $V(G)$ is partitioned into classes $L_1, \ldots, L_t$ and a set I is independent if $|I \cap L_i| \leq 1$ for every label $i \in \{1, \ldots, t\}$. In this way, a path of G of rank at least k is a path containing vertices of at least k different (color) classes among $L_1, \ldots, L_t$.

Randomized FPT algorithms for Maximum Rank (s,t)-Path. The parameterized complexity of Maximum Rank (s,t)-Path was unknown until very recently. The first FPT algorithm for Maximum Rank (s,t)-Path was given in [15]. This algorithm runs in time $2^{\mathcal{O}(k^2 \log(q+k))} n^{\mathcal{O}(1)}$ and works on frameworks with matroids represented in finite fields of order q. Also, Eiben, Koana, and Wahlström [13], using different techniques, obtained an FPT algorithm for the same problem that runs in time $2^k n^{\mathcal{O}(1)}$ on frameworks with matroids representable over fields of characteristic two. These two algorithms use two different algebraic methods. The algorithm of [15] extends the celebrated algebraic technique based on *cancellation of monomials* used by Björklund, Husfeldt, and Taslaman [5] to solve the T-Cycle problem, while the algorithm of [13] utilizes the toolbox of *(constrained) multilinear detection* [3, 4, 34, 35] combined with *determinantal sieving* [13]. Both these algorithms involve polynomial identity testing and invoke the Schwartz-Zippel lemma, and therefore

are randomized. In fact, because of the crucial use of the Schwartz-Zippel lemma in both these algorithms, as the authors of [13] state it, "derandomization appears infeasible" for the algorithms of [15] and [13] for Maximum Rank (s,t)-Path. Therefore, the next challenge is to obtain *derandomized* FPT algorithms for this problem.

Our results. Our main result establishes the first *deterministic* FPT algorithm for Maximum Rank (s,t)-Path on frameworks of planar graphs and matroids representable over finite fields or over the field of rationals. We use $|G|$ to denote the number of vertices of a graph G and $\|M\|$ to denote the bit-length of the representation matrix of a linear matroid M.

▶ **Theorem 1.** *There is a deterministic algorithm that, given a framework (G,M), where G is a planar graph G and M is represented as a matrix over a finite field or over $\mathbb{Q}$, two vertices $s,t \in V(G)$ and an integer k, in time $2^{2^{\mathcal{O}(k \log k)}} \cdot (|G| + \|M\|)^{\mathcal{O}(1)}$ either returns an (s,t)-path of G of rank at least k, or determines that G has no such (s,t)-path.*

Note that the randomized FPT algorithms of [15] and [13] work for matroids representable over finite fields or fields of characteristic two. The algorithm of Theorem 1, apart from being the first deterministic algorithm for Maximum Rank (s,t)-Path, is also the first FPT algorithm for frameworks whose matroids are *not* represented over a finite field or a field of characteristic two, but are represented over $\mathbb{Q}$.

Our techniques. To design the deterministic FPT algorithm of Theorem 1, we follow a different proof strategy than that of [15] and [13]. Our approach is based on the *win/win* arguments of the celebrated *irrelevant vertex technique* of Robertson and Seymour [42]. The general scheme of this technique is the following. If the graph satisfies certain combinatorial properties, then one can identify a vertex of the graph that can be declared *irrelevant*, meaning that its deletion results in an equivalent instance of the problem. Therefore, after deleting this vertex, we can iterate on the (equivalent) reduced instance. Once this reduction rule can not be further applied, the obtained reduced instance is equivalent to the original one and also "simpler". Therefore, one remains to argue that the problem can be solved efficiently in the reduced equivalent instance. This is a standard technique in parameterized algorithms design – see, for example, [2, 17, 19, 20, 22–30, 32, 40, 44] (see also [11, Section 7.8]). The standard mesure of complexity of instances for the application of the irrelevant vertex technique is *treewidth*. In particular, the strategy is formulated as follows. As long as the treewidth of the instance is large enough, detect and remove irrelevant vertices. If the treewidth is small, then solve the problem on this equivalent instance using dynamic programming.

Our application of the irrelevant vertex technique is inspired by the algorithm of Kawarabayashi [25] for T-cycle and extends its methods. In a typical irrelevant-vertex argument, one has to prove that every solution can "avoid" a vertex that will be declared irrelevant. For example, in the classical application of Robertson and Seymour [42] for the Disjoint Paths problem, one should argue that (if the graph has large treewidth) any collection of disjoint paths between certain terminals can be "rerouted away" from a vertex v and this vertex should be declared irrelevant. In our case, where we seek an (s,t)-path of *large rank* in a framework, this rerouting should guarantee that large rank is preserved. In general, to deal with such problems on frameworks, one should employ new arguments to adjust this technique to take into account the structure of the matroid. The way we circumvent this problem for Maximum Rank (s,t)-Path is to formulate such a rerouting argument in a "sufficiently insulated" area of the graph where independent sets of the matroid M

appear in a homogeneous way. Planarity of the input graph allows to find such an area using the grid-like structure of *walls*. An overview of this approach is provided in Subsection 1.1. This application of the irrelevant vertex technique for frameworks is novel and illustrates an interesting interplay between combinatorial structures and algebraic properties, that may be of independent interest.

The dynamic programming on graphs of bounded treewidth is pretty standard (see, e.g., [12]) up to one detail. To encode a partial solution, we keep the information about vertices forming independent sets of matroid M visited by a partial solution. However, the number of independent sets of size at most k in M could be of order n^k. Thus a naive encoding of partial solutions would result in blowing-up of the computational complexity. To avoid this, we store only *representative* sets (see [18, 36]) instead of all possible independent sets. Both randomized [18] and deterministic [36] constructions of representative sets require a linear representation of M. This is the reason why Theorem 1 is stated for linear matroids. We point out that the dynamic programming subroutine for graphs of bounded treewidth is the only place in the proof of Theorem 1 requiring a representation of M. It is an interesting open question, whether Maximum Rank (s,t)-Path is FPT when parameterized by k and the treewidth if the input matroid is given by its independence oracle.

1.1 Overview of the proof of Theorem 1

Our general approach is the following. We show that if the treewidth of the input graph G is $2^{\mathcal{O}(k \log k)}$, then Maximum Rank (s,t)-Path can be solved in FPT time by a dynamic programming algorithm. Otherwise, if the treewidth is sufficiently large, we give an algorithm that either finds an (s,t)-path of rank at least k or identifies an *irrelevant* vertex v, that is, a vertex whose deletion results in an equivalent instance of the problem. In the latter case, we delete v and iterate on the reduced instance.

If the treewidth of the input graph is large, i.e., of order $2^{\Omega(k \log k)}$, we exploit the grid-minor theorem of Robertson and Seymour for planar graphs [43] that asserts that a planar graph either contains $(w \times w)$-grid as a minor or the treewidth is $\mathcal{O}(w)$. More precisely, we have that given a plane embedding of G, we can find a plane h-wall for $h = 2^{\Omega(k \log k)}$ as a topological minor or, equivalently, a plane subgraph of G that is a subdivision of such a wall. To explain our arguments, we need some notions that are informally explained here by making use of figures. In particular, an example of an h-wall for $h = 7$ is given in Figure 1.

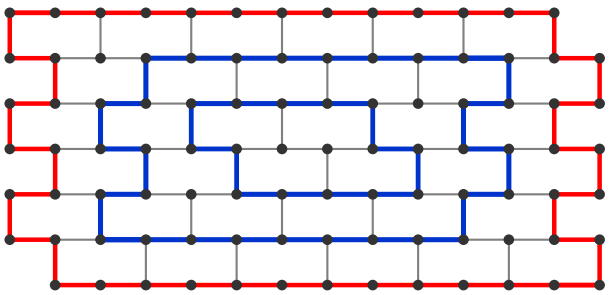


Figure 1 A 7-wall and its layers.

Note that an h-wall has $\lfloor h/2 \rfloor$ nested cycles, called *layers*, that are shown in Figure 1 in red and blue. The layer forming the boundary of a wall is called the *perimeter* of the wall and is shown in red in the figure. We extend the notions of layers and perimeter for a *subdivided* h-wall, that is, the graph obtained from an h-wall by replacing some of its

edges by paths. The vertices of the initial h-wall, i.e., before replacing edges by paths, are called the *branch vertices* of the subdivided h-wall. Given a plane subdivided h-wall W in G, we call the subgraph of G induced by the vertices on the perimeter and inside the inner face of the perimeter the *compass* of W and denote it by $\mathsf{compass}(W)$. Notice that we can assume that the compass of the subdivided h-wall W in G does not contain the terminal vertices s and t by switching to a smaller subwall if necessary. Furthermore, we can assume that $\mathsf{compass}(W)$ is a 2-connected graph as any (s,t)-path can only contain vertices of the biconnected component of $\mathsf{compass}(W)$ containing W. Also we can assume that G has two disjoint paths connecting s and t with two distinct vertices on the perimeter of W; otherwise, any vertex of $\mathsf{compass}(W)$ outside the perimeter is trivially irrelevant.

Observe that for any nontrivial subwall W' of W, $\mathsf{compass}(W')$ is also 2-connected. Therefore, for every two distinct vertices x and y on the perimeter of W' and any $z \in V(\mathsf{compass}(W'))$, $\mathsf{compass}(W')$ has internally disjoint (x,z) and (y,z)-paths. In particular, given a set of vertices $S \subseteq V(\mathsf{compass}(W'))$ that are independent with respect to M, we can join any $z \in S$ with x and y by disjoint paths. This observation is crucial for us.

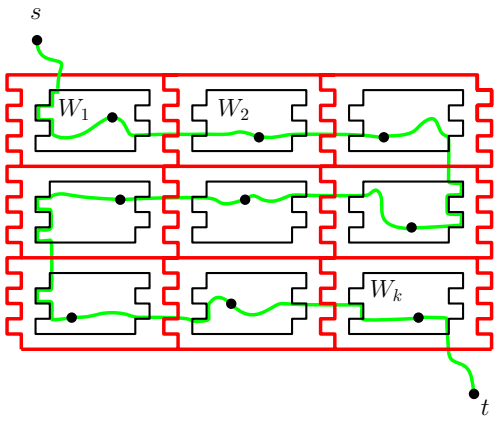


Figure 2 An (s,t)-path for walls of big rank.

Suppose that there is a packing of k subwalls $W_1,\dots,W_k$ in W separated by paths in W as it is shown in Figure 2 such that the rank $r(\mathsf{compass}(W_i)) \geq k$ for $i \in \{1,\dots,k\}$. Then we can choose vertices $v_1,\dots,v_k$ in $\mathsf{compass}(W_1),\dots,\mathsf{compass}(W_k)$, respectively, in such a way that $\{v_1,\dots,v_k\}$ is an independent set of M. Then by our observation, we can construct an (s,t)-path in G that goes through $v_1,\dots,v_k$ as it is shown in the figure in green. Suppose that this is not the case. Then, by zooming inside the wall, we can assume that $r(\mathsf{compass}(W)) < k$. Moreover, by recursive zooming, we can find a subwall W' of W with the following structural properties (see Figure 3).

- There is a packing of $k+1$ subwalls $W_0, W_1, \dots, W_k$ in W' separated by paths in W' shown in red in Figure 3 such that $r(\mathsf{compass}(W_i)) = r(\mathsf{compass}(W'))$ for $i \in \{0,\dots,k\}$.
- The packing of $W_0, W_1, \dots, W_k$ is surrounded by $\mathcal{O}(k^2)$ "insulation" layers of W' shown in blue.

We claim that vertices of W_0 are irrelevant.

To see this, consider an (s,t)-path P of rank at least k in G. We show that if P goes through a vertex of W_0, then the path can be rerouted as it is shown in Figure 3 in green to avoid W_0. Consider an independent set $X \subseteq V(P)$ of rank k and let $u_1,\dots,u_\ell$ be the vertices of X that are not spanned by $V(\mathsf{compass}(W'))$ in M. Then $u_1,\dots,u_\ell$ are outside W'. We prove that there are two distinct vertices x and y on the inner insulation layer of W',

and an (s,x)-path P_1 and an (y,t)-path P_2 such that (i) x and y are unique vertices of these paths in the inner insulation layer, and (ii) $u_1,\dots,u_\ell \in V(P_1)\cup V(P_2)$. The proof that $\mathcal{O}(k^2)$ insulation layers are sufficient for rerouting P is non-trivial. In particular, we adapt the ideas from [25] as well as the structural results of Kleinberg [31]. Further, using the fact that $r(\mathsf{compass}(W_i)) = r(\mathsf{compass}(W'))$ for $i \in \{1,\dots,k\}$, we show that for every independent set I' of M consisting of vertices in $\mathsf{compass}(W')$, one can also find an independent set I_i of M in $\mathsf{compass}(W_i)$ such that $|I_i| = |I'|$, for every $i \in \{1,\dots,k\}$. Therefore, one can select, for every W_i, a vertex $v_i \in I_i$ and this choice can be made so that $r(\{v_1,\dots,v_k\}) = r(\mathsf{compass}(W'))$. Then we construct an (x,y)-path Q in the inner part of W' such that (i) Q is internally disjoint with P_1 and P_2, (ii) Q goes through $v_1,\dots,v_k$, and (iii) Q avoids W_0. We have that $P' = P_1QP_2$ is an (s,t)-path that goes through $u_1,\dots,u_\ell$ and $v_1,\dots,v_k$. Note that, replacing the vertices of X that are spanned by $V(\mathsf{compass}(W'))$ by the vertices $\{v_1,\dots,v_k\}$, we obtain the set $X' = \{u_1,\dots,u_\ell,v_1,\dots,v_k\}$ and $r(X') = r(X)$, and the latter holds since $r(\{v_1,\dots,v_k\}) = r(\mathsf{compass}(W'))$. Therefore $r(P') \geq r(X) \geq k$. Since Q avoids W_0, P' has the same property.

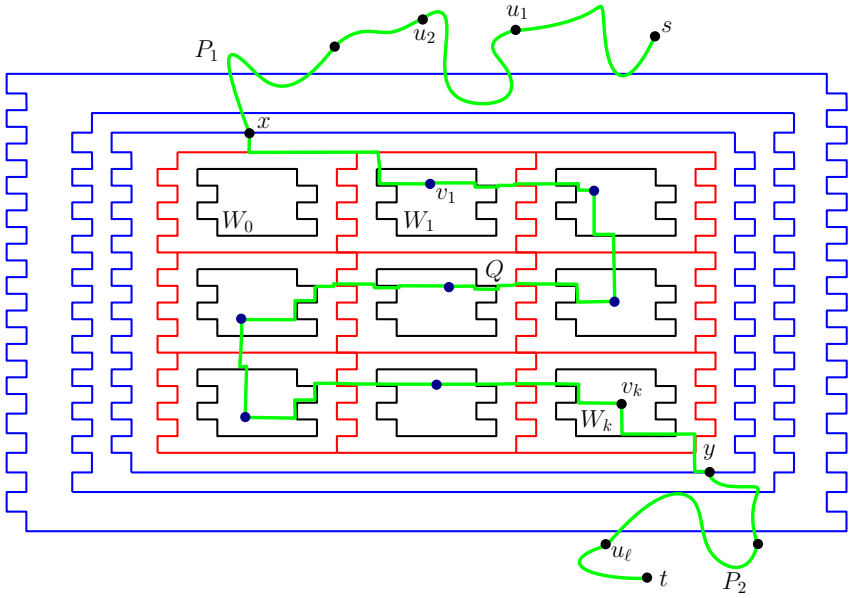


Figure 3 Rerouting an (s,t)-path.

Finally, we note that the algorithm of Kawarabayashi [25] for T-CYCLE works for general graphs. The statement of Theorem 1 is limited to planar graphs and planarity is required to ensure that the rerouting does not decrease the rank of an (s,t)-path. It is quite plausible that with additional technicalities our method could be lifted when the underlying graph of the framework is of bounded genus, and more generally, minor-free. However, it is very unclear, whether rerouting that does not decrease the rank could be achieved for general graphs. It remains the main obstacle towards pushing the irrelevant vertex technique from frameworks with planar graphs to frameworks with general graphs.

Organization of the paper. In Section 2 we show how to reduce to instances of bounded treewidth using the irrelevant vertex technique. Results whose proofs are omitted in this extended abstract are marked with a star ($\star$) and their proofs can be found in the full version [16]. We also refer to the full version for formal definitions of the aforementioned notions. We conclude in Section 3 with open questions and possible future research directions.

2 Rerouting paths and cycles

In this section, our goal is to prove Theorem 1 that we restate here.

▶ **Theorem 1.** *There is a deterministic algorithm that, given a framework (G, M), where G is a planar graph G and M is represented as a matrix over a finite field or over $\mathbb{Q}$, two vertices $s, t \in V(G)$ and an integer k, in time $2^{2^{\mathcal{O}(k \log k)}} \cdot (|G| + \|M\|)^{\mathcal{O}(1)}$ either returns an (s,t)-path of G of rank at least k, or determines that G has no such (s,t)-path.*

The algorithm of Theorem 1 consists of two parts. In the first part, we use the irrelevant vertex technique in order to design an algorithm that removes vertices form the input graph as long as its treewidth is big enough. In order to do this, we show a combinatorial result that allows us to argue that, given a planar graph and a wall of it and a vertex set S that lies outside the wall, if there is a path P that contains S and invades deeply enough inside the wall, we can find another path P' that contains S (with the same endpoints as P) and avoids some "central area" of the wall (Lemma 4). Then, we give an algorithm (Lemma 6) that given a planar graph of "big enough" (as a function of k) treewidth, outputs, in time $2^{2^{\mathcal{O}(k \log k)}} \cdot (|G| + \|M\|)^{\mathcal{O}(1)}$, either a path of rank at least k or an irrelevant vertex. The dynamic programming algorithm that solves the problem in graphs of bounded treewidth is included in the full version of the paper.

2.1 Rerouting paths and cycles

In this subsection, we aim to prove the main combinatorial result (Lemma 4) that allows us to find an (s,t)-path that contains a given set S and avoids some inner part of a given wall. Before stating Lemma 4, we state the following result (Lemma 3) that will be an important tool for the proof of Lemma 4. The proof of Lemma 3 is inspired by the proof of [25, Lemma 1]. An *in-peg* of the perimeter of a wall W is a vertex on the perimeter of W that has degree three in W.

▶ **Lemma 3** ($\star$)**.** *Let G be a planar graph, let $k \in \mathbb{N}$, let W be a wall of height at least $2k+3$, and let $s, t \in V(G) \setminus V(\mathsf{compass}(W))$. Also, let $E = \{e_1, \ldots, e_k, e_{k+1}, e_{k+2}\}$ be a set of $k+2$ edges of G, where, for every $i \in \{1, \ldots, k\}$, $e_i = \{v_i, u_i\}$, $e_{k+1} = \{v_{k+1}, s\}$, $e_{k+2} = \{v_{k+2}, t\}$, and let X be the set $\{v_{k+1}, v_{k+2}\} \cup \bigcup_{i \in \{1,\ldots,k\}} \{v_i, u_i\}$. If every $v \in X$ is an in-peg of $\mathsf{perim}(W)$, then there is an (s,t)-path in G that contains the edges $e_1, \ldots, e_{k+2}$ and its intersection with $\mathsf{compass}(W^{(k+1)})$ is a path of $\mathsf{perim}(W^{(k+1)})$ whose endpoints are branch vertices of W.*

We now prove the following result.

▶ **Lemma 4.** *There is a function $h : \mathbb{N} \to \mathbb{N}$ such that if $k, z \in \mathbb{N}$, G is a planar graph, $s, t \in V(G)$, S is a subset of $V(G)$ of size at most k, W is a wall of G of at least $h(k)$ layers and whose compass is disjoint from $S \cup \{s,t\}$, and P is an (s,t)-path of G such that $S \subseteq V(P)$ and P intersects $V(\mathsf{inn}(W^{(h(k))}))$, then there is an (s,t)-path $\tilde{P}$ of G such that $S \subseteq V(\tilde{P})$ and its intersection with $\mathsf{compass}(W^{(h(k))})$ is a path of $\mathsf{perim}(W^{(h(k))})$ whose endpoints are branch vertices of W. Moreover, $h(k) = \mathcal{O}(k^2)$.*

Proof. We set $h(k) := 2k \cdot (k+2) + 2k + 1$. Let W be a wall of at least $h(k)$ layers. For $i \in \{1, \ldots, k+2\}$, we use C_i to denote the layer $L_{2k \cdot (i-1)+1}$ of W. Intuitively, we take C_1 to be the first layer of W and for every $i \in \{2, \ldots, k+2\}$, we take C_i to be the $2k$-th consecutive layer after C_{i-1}. Also, we use D_i to denote the vertex set of $\mathsf{compass}(W^{(2k \cdot (i-1)+1)})$. Keep in mind that C_i is the perimeter of $W^{(2k \cdot (i-1)+1)}$. For every $i \in [k+2]$, we consider the collection $\mathcal{F}_i$ of paths of G that are subpaths of P that intersect D_i only on their endpoints

and that there is an onto function mapping each vertex $u \in S \cup \{s,t\}$ to the path in $\mathcal{F}_i$ that contains u. Intuitively, for each $u \in S \cup \{s,t\}$ we consider the maximal subpath of P that contains u and intersects D_i only on its endpoints and we define $\mathcal{F}_i$ to be the collection of these maximal paths (see Figure 4 for an example).

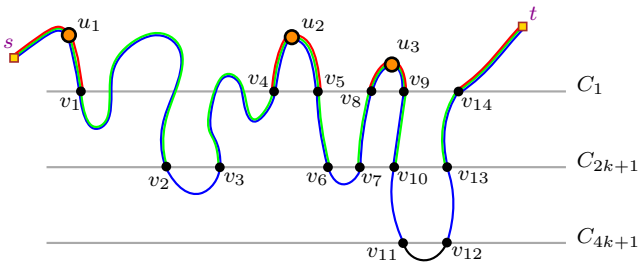


Figure 4 An example of an (s,t)-path P containing an independent set $S = \{u_1, u_2, u_3\}$. In this example, $\mathcal{F}_1$ is the collection of the four red paths (the ones with endpoints (s, v_1), (v_4, v_5), (v_8, v_9), and (v_{14}, t)), $\mathcal{F}_2$ is the collection of the four green paths (the ones with endpoints (s, v_2), (v_3, v_6), (v_7, v_{10}), (v_{13}, t)), and $\mathcal{F}_3$ is the collection of the two blue paths (the (s, v_{11})-path and the (v_{12}, t)-path).

Observe that $|\mathcal{F}_1| \leq k+2$ (since $|S \cup \{s,t\}| \leq k+2$) and $|\mathcal{F}_{k+2}| \geq 2$ (since $V(P) \cap V(\mathsf{inn}(W^{(h(k))})) \neq \emptyset$ and therefore P intersects at least twice every C_i, $i \in [k+2]$). For every $i \in \{1, \ldots, k+2\}$, we assume that $\mathcal{F}_i = \{F_{i,1}, \ldots, F_{i,|\mathcal{F}_i|}\}$, where the ordering is given by traversing P from s to t. For every $i \in \{1, \ldots, k+2\}$, we set $\mathcal{Q}_i = \{Q_{i,1}, \ldots, Q_{i,|\mathcal{F}_i|-1}\}$, where, for each $j \in [|\mathcal{F}_i| - 1]$, $Q_{i,j}$ is the minimal subpath of P that intersects both $V(F_{i,j})$ and $V(F_{i,j+1})$. Observe that, for every $i \in \{1, \ldots, k+2\}$, P is the concatenation of the paths $F_{i,1}, Q_{i,1}, F_{i,2}, \ldots, Q_{i,|\mathcal{F}_i|-1}, F_{i,|\mathcal{F}_i|}$. In Figure 4, $\mathcal{Q}_1 = \{Q_{1,1}, Q_{1,2}, Q_{1,3}\}$, where $Q_{1,1}$ is the (v_1, v_4)-subpath, $Q_{1,2}$ is the (v_5, v_8)-subpath, and $Q_{1,3}$ is the (v_9, v_{14})-subpath of P, $\mathcal{Q}_2 = \{Q_{2,1}, Q_{2,2}, Q_{2,3}\}$, where $Q_{2,1}$ is the (v_2, v_3)-subpath, $Q_{2,2}$ is the (v_6, v_7)-subpath, and $Q_{2,3}$ is the (v_{10}, v_{13})-subpath of P, and $\mathcal{Q}_3$ consists of the (v_{11}, v_{12})-subpath $Q_{3,1}$ of P.

It is easy to see that for every $i \in \{1, \ldots, k+1\}$, $|\mathcal{F}_{i+1}|$ is equal to $|\mathcal{F}_i|$ minus the number of paths in $\mathcal{Q}_i$ that do not intersect C_{i+1} and therefore, $|\mathcal{F}_i| \geq |\mathcal{F}_{i+1}|$. Therefore, given that $|\mathcal{F}_1| \leq k+2$, $|\mathcal{F}_{k+2}| \geq 2$, and for every $i \in \{1, \ldots, k+1\}$, $|\mathcal{F}_i| \geq |\mathcal{F}_{i+1}|$, there is an $i_0 \in \{1, \ldots, k+1\}$ such that $|\mathcal{F}_{i_0}| = |\mathcal{F}_{i_0+1}|$ (if there are many such i_0, we pick the minimal one). This implies that every path in $\mathcal{Q}_{i_0}$ intersects C_{i_0+1}.

For each $F \in \mathcal{F}_{i_0}$, we denote by v_F and u_F the endpoints of F. We define the graph G' obtained from G after removing the internal vertices of every $F \in \mathcal{F}_{i_0}$ (i.e., the vertex set $\bigcup_{F \in \mathcal{F}_{i_0}} (V(F) \setminus \{v_F, u_F\})$) and adding the edge $\{v_F, u_F\}$ for every $F \in \mathcal{F}_{i_0}$. Observe that G' is also planar and contains D_{i_0} as a subgraph. Moreover, notice that, for every $F \in \mathcal{F}_{i_0}$, $\{v_F, u_F\} \in V(C_{i_0}) \cup \{s,t\}$. In Figure 4, $|\mathcal{F}_1| = |\mathcal{F}_2|$ and thus G' is obtained after replacing each 3-colored path with an edge.

In the rest of the proof we will argue that, in G', there is an (s,t)-path that contains all edges $\{v_F, u_F\}$, $F \in \mathcal{F}_{i_0}$, and its intersection with $V(\mathsf{compass}(W^{(h(k))}))$ is the vertex set of a subdivided edge of W that lies in $\mathsf{perim}(W^{(h(k))})$. Having such a path in hand, we can replace each edge $\{v_F, u_F\}$, $F \in \mathcal{F}_{i_0}$ with the corresponding path F and thus obtain the path $\tilde{P}$ claimed in the statement of the lemma.

We will denote by C the cycle C_{i_0} (that is the layer $L_{2k\cdot(i_0-1)+1}$) and by C' the layer $L_{2k\cdot i_0}$. To get some intuition, recall that $C_{i_0+1} = L_{2k\cdot i_0+1}$ and therefore C' is the layer of W "preceding" C_{i_0+1}. Since every path in $\mathcal{Q}_{i_0}$ intersects C_{i_0+1}, it holds that every path in $\mathcal{Q}_{i_0}$ intersects C' at least twice. Therefore, if we set $Y := V(C) \cap \bigcup_{F \in \mathcal{F}_{i_0}} \{v_F, u_F\}$ and $\ell := |Y|$, then $\ell \leq 2k$ and there are ℓ disjoint paths from Y to C' (for an example, see the left part of Figure 5).

Recall that $\mathsf{perim}(W^{(2k\cdot i_0)}) = C'$. We set B to be the set of branch vertices of W that are in $V(C')$ and have degree three in $W^{(2k\cdot i_0)}$. Also, we set $\mathcal{K}$ to be the graph $G' \setminus V(\mathsf{inn}(W^{(2k\cdot i_0)}))$. The next claim states that there also exist ℓ disjoint paths from Y to B in $\mathcal{K}$. We omit the proof and we refer the reader to the full version [16].

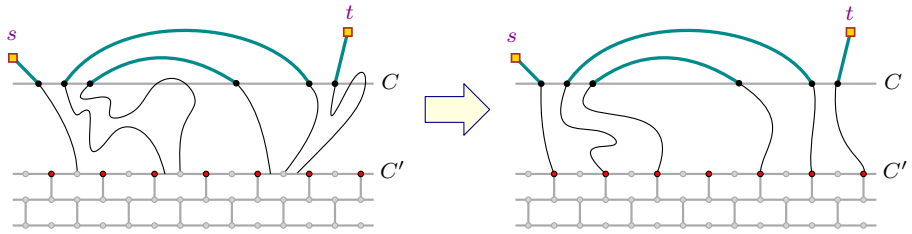


Figure 5 A visualization of the statement of Claim 5. In both figures, the edges $\{v_F, u_F\}$ are depicted in blue, the black vertices correspond to the set Y and the red vertices correspond to the set B. In the left figure, we illustrate $|Y|$ disjoint paths from Y to C', while in the right figure, we illustrate $|Y|$ disjoint paths from Y to B.

▷ Claim 5 ($\star$). There is a set $X \subseteq B$, a bijection $\rho : Y \to X$, and a collection $\mathcal{P} = \{P_v \mid v \in Y\}$ of pairwise disjoint paths where, for every $v \in Y$, P_v is a $(v, \rho(v))$-path in $\mathcal{K}$.

Following Claim 5, let $X \subseteq B$, let a bijection $\rho : Y \to X$, and let a collection $\mathcal{P} = \{P_v \mid v \in Y\}$ of pairwise disjoint paths such that for every $v \in Y$, P_v is a $(v, \rho(v))$-path in $\mathcal{K}$.

Now, for each $F \in \mathcal{F}_{i_0}$, we consider the path P_F obtained after joining the paths P_{v_F} and P_{u_F} by the edge $\{v_F, u_F\}$ (in the case where $s, t \in \{v_F, u_F\}$, we just extend the corresponding path in $\mathcal{P}$ by adding the edge $\{v_F, u_F\}$). Let G'' be the graph obtained from G' after contracting each $P_F, F \in \mathcal{F}_{i_0}$ to an edge e_{P_F} and let $E = \{e_{P_F} \mid F \in \mathcal{F}_{i_0}\}$. Then, notice that G'' contains $W^{(2k\cdot i_0)}$ as a subgraph and since $h(k) = 2k \cdot (k+2) + 2k + 1$, the wall $W^{(2k\cdot i_0)}$ has at least $k+1$ layers and therefore height at least $2k+3$. Therefore, by Lemma 3, G'' contains an (s,t)-path that contains all edges in E and its intersection with $\mathsf{compass}(W^{(2k\cdot i_0+k+1)})$ is a path of $\mathsf{perim}(W^{(2k\cdot i_0+k+1)})$ whose endpoints are branch vertices of W.

Thus, using this (s,t)-path in G'', we can find an (s,t)-path $P^\star$ in G that contains S and its intersection with $\mathsf{compass}(W^{(2k\cdot i_0+k+1)})$ is a path of $\mathsf{perim}(W^{(2k\cdot i_0+k+1)})$ whose endpoints, say x and y, are branch vertices of W. Finally, let an (x,y)-path $R_{x,y}$ in $\mathsf{compass}(W^{(2k\cdot i_0+k+1)})$ whose intersection with $\mathsf{compass}(W^{(h(k))})$ is a path of $\mathsf{perim}(W^{(h(k))})$ whose endpoints are branch vertices of W. The proof concludes by observing that $(P^\star \setminus V(\mathsf{compass}(W^{(2k\cdot i_0+k+1)})))\cup R_{x,y}$ is the (s,t)-path claimed in the statement of the lemma. ◀

We stress that, while Lemma 4 deals with the case of "rerouting" an (s,t)-path, we can apply the same arguments to "reroute" a cycle that contains a fixed set S away from the inner part of some wall.

2.2 Equivalent instances of small treewidth

In this subsection, we design an algorithm that receives a framework (G, M), where G is a planar graph of "big enough" treewidth, and two vertices $s, t \in V(G)$, and outputs either a report that G contains an (s,t)-path of rank at least k, or an irrelevant vertex that can be safely removed. In frameworks, to remove a vertex, one has to remove this vertex from G and also restrict the matroid.

Restrictions of matroids. Let $M = (V, \mathcal{I})$ be a matroid and let $S \subseteq V$. We define the *restriction of M to S*, denoted by $M|S$, to be the matroid on the set S whose independent sets are the sets in $\mathcal{I}$ that are subsets of S. Given a $v \in V$, we denote by $M \setminus v$ the matroid $M|(V \setminus \{v\})$.

The goal of this subsection is to prove the following.

▶ **Lemma 6.** *There is a function $g : \mathbb{N} \to \mathbb{N}$ and an algorithm that, given an integer $k \in \mathbb{N}$, a framework (G, M), where M is a matroid for which we can verify independence in time $\|M\|^{\mathcal{O}(1)}$, and G is a planar graph of treewidth at least $g(k)$, and two vertices $s, t \in V(G)$, outputs, in time $2^{2^{\mathcal{O}(k \log k)}} \cdot (|G| + \|M\|)^{\mathcal{O}(1)}$,*

- *either a report that G contains an (s,t)-path of rank at least k, or*
- *a vertex $v \in V(G)$ such that (G, M, k, s, t) and $(G \setminus v, M \setminus v, k, s, t)$ are equivalent instances of* Maximum Rank (s,t)-Path.

Moreover, $g(k) = 2^{\mathcal{O}(k \log k)}$.

Keep in mind that, if M is represented over a finite field or $\mathbb{Q}$, we can verify independence in time that is a polynomial in $\|M\|$. In order to prove Lemma 6, we need some additional definitions and results.

Packings of walls. Let G be a planar graph and W be a wall of G. Let $z, r \in \mathbb{N}$ and let q be a non-negative odd integer. We say that W *admits an (z, r, q)-packing of walls*, if W has height at least h, for some odd $h \geq 2z$, and there is a collection $\mathcal{W} = \{W_0, W_1, \ldots, W_{r-1}\}$ of subwalls of W, such that for every $i \in \{0, \ldots, r-1\}$, W_i is a subwall of W of height at least q such that $V(W_i)$ is a subset of $V(W^{(z+1)})$, and for every $i, j \in \{0, \ldots, r-1\}$, with $i \neq j$, $V(\mathsf{compass}(W_i))$ and $V(\mathsf{compass}(W_j))$ are disjoint. We call $\mathcal{W}$ an *(z, r, q)-packing* of W (see Figure 3 for a visualization of a packing of a wall W).

▶ **Observation 7.** *Given $z, r \in \mathbb{N}$, an odd integer $q \in \mathbb{N}$, and a planar graph G, every wall W of G of height at least $2z + \lceil \sqrt{r} \rceil \cdot (q+1) + 1$ admits a (z, r, q)-packing.*

Let W be a wall of a planar graph. We use $\rho(W)$ to denote $r(V(\mathsf{compass}(W)))$.

▶ **Lemma 8 ($\star$).** *There is a function $f : \mathbb{N}^4 \to \mathbb{N}$ and an algorithm that, given integers $k, z, r, q \in \mathbb{N}$, where q is odd, a framework (G, M), where G is planar and M is a matroid for which we can verify independence in time $\|M\|^{\mathcal{O}(1)}$, and a wall W of G of height at least $f(k, z, r, q)$ such that $\rho(W) \leq k$, outputs, in $(k+1) \cdot r \cdot (|G| + \|M\|)^{\mathcal{O}(1)}$ time, a subwall W' of W of height h, for some odd $h \in \mathbb{N}$ such that $h \geq 2z$, and a (z, r, q)-packing $\mathcal{W}$ of W' such that for every $W_i \in \mathcal{W}$, $\rho(W_i) = \rho(W')$. Moreover, $f(k, z, r, q) = \mathcal{O}(r^{k/2} \cdot z \cdot q)$.*

We are now ready to prove Lemma 6.

Proof of Lemma 6. We set $b = h(k)$, $x = k + 1$, $z = (k+1) \cdot b$, $q = f(k-1, z, x, 3)$, $r = \lceil \sqrt{k} \rceil \cdot (q+1) + 3$, and $g(k) = 36(r+1)$. We first assume that G is 2-connected. If G is not connected, then we break the problem in subproblems, each one corresponding to a 2-connected component B of G and if the vertices of B are separated from s or t by a cut-vertex v of G, then we consider the problem where v is set to be s or t, respectively.

Since the treewidth of G is at least $g(k) = 36(r+1)$, due to [21, Lemma 4.2], G has a $(4r+1)$-wall. We then consider an r-wall W of G such that $s, t \notin V(\mathsf{compass}(W))$ and an $(1, k, q)$-packing $\tilde{\mathcal{W}} = \{\tilde{W}_1, \ldots, \tilde{W}_k\}$ of W. This $(1, k, q)$-packing exists because of the fact that $r = \lceil \sqrt{k} \rceil \cdot (q+1) + 3$ and due to Observation 7 and we can find it in $\mathcal{O}(n)$ time. For every $i \in \{1, \ldots, k\}$, we set $K_i := V(\mathsf{compass}(\tilde{W}_i))$. Then, compute the rank of K_i, for each $i \in \{1, \ldots, k\}$. This can be done in time $k \cdot (|G| + \|M\|)^{\mathcal{O}(1)}$.

If every K_i has rank at least k, then notice that there is a set $S \subseteq V(G)$ such that $r(S) = k$ and for every $i \in \{1, \ldots, k\}$, $|S \cap K_i| = 1$. To obtain an (s,t)-path P such that $S \subseteq V(P)$, we do the following: We first pick two disjoint paths P_s, P_t from the perimeter of W to s and t respectively (these exist since G is 2-connected). Let D be the perimeter of W and let s' and t' be the endpoints of P_s and P_t in D. Also, let L_2 be the second layer of W. Observe that, since the compass of a wall is a connected graph, there is also a path $\overline{P}$ in G such that the endpoints, say x, y, of $\overline{P}$ are in L_2, no internal vertex of $\overline{P}$ is a vertex of L_2, and $S \subseteq V(\overline{P})$. Finally, observe that there exist two disjoint paths $P_{s'x}, P_{t'y}$ in the closed disk bounded by D and L_2 connecting s' with x and t' with y, respectively, and that $P := P_s \cup P_{s'x} \cup \overline{P} \cup P_{t'y} \cup P_t$ is an (s,t)-path such that $S \subseteq V(P)$ (see Figure 2).

Suppose now that there is an $i \in \{1, \ldots, k\}$ such that the rank of K_i is at most $k-1$. Since the corresponding wall $\tilde{W}_i$ is of height at least $q = f(k-1, z, x, 3)$, by Lemma 8, we can find a subwall W' of $\tilde{W}_i$ of height h, for some odd $h \geq 2z$ and a $(z, k+1, 3)$-packing $\mathcal{W} = \{W_0, W_1, \ldots, W_k\}$ of W', so that for every $i \in \{0, \ldots, k\}$, $\rho(W_i) = \rho(W')$. Let v be a central vertex of W_0.

We now prove that (G, M, k, s, t) and $(G \setminus v, M \setminus v, k, s, t)$ are equivalent instances of Maximum Rank (s,t)-Path. We show that if (G, M, k, s, t) is a yes-instance, then $(G \setminus v, M \setminus v, k, s, t)$ is also a yes-instance, since the other implication is trivial. If (G, M, k, s, t) is a yes-instance, then there is a set of vertices $S = \{u_1, \ldots, u_k\} \subseteq V(G)$ and an (s,t)-path P in G such that $r(S) = k$ and $S \subseteq V(P)$. The fact that $z = (k+1) \cdot b$ implies that there is an $i \in \{1, \ldots, k+1\}$ such that the vertex set $V(\mathsf{compass}(W'^{((i-1)\cdot b+1)}) \setminus V(\mathsf{inn}(W'^{(i\cdot b)})))$, which we denote by D_i, does not intersect S. Let S_{in} be the vertices of S that are contained in $\mathsf{compass}(W'^{(i\cdot b)})$ and let S_{out} be the set $S \setminus S_{\text{in}}$. We will show that there is a set $S' \in \mathcal{I}(M \setminus v)$ and a path P' such that $r(S_{\text{out}} \cup S') \geq k$, $S_{\text{out}} \cup S' \subseteq V(P')$ and $V(P') \subseteq V(G \setminus v)$.

We assume that $v \in V(P)$, otherwise we set $S' := S_{\text{in}}$ and $P' := P$ and the lemma follows. By Lemma 4, there is a path $\tilde{P}$ such that $S_{\text{out}} \subseteq V(\tilde{P})$ and $V(\tilde{P}) \cap V(\mathsf{compass}(W'^{(i\cdot b)}))$ is the vertex set of a path $\hat{P}$ of W'_0 that lies in $\mathsf{perim}(W'^{(i\cdot b)})$ and whose endpoints are branch vertices of $W'^{(i\cdot b)}$. Let $s_{\hat{P}}$ and $t_{\hat{P}}$ be the endpoints of $\hat{P}$.

We can assume that $\rho(W_i) = \rho(W') > 0$, for every $i \in \{0, \ldots, k\}$, since otherwise $S_{\text{in}} = \emptyset$ and the claim holds trivially. For every $i \in \{0, \ldots, k\}$, since $\rho(W_i) = \rho(W')$ and S_{in} is an independent set of M that is a subset of $\mathsf{compass}(W')$, there is an independent set $S_i \subseteq V(\mathsf{compass}(W_i))$ such that $|S_i| = |S_{\text{in}}|$. Furthermore, because $\rho(W_i) = \rho(W')$ for $i \in \{0, \ldots, k\}$, we can choose a set $S' = \{v_1, \ldots, v_k\}$ where v_i is a vertex in S_i for $i \in \{1, \ldots, k\}$ in such a way that $r(S') = \rho(W')$. Then $r(S_{\text{out}} \cup S') = |S_{\text{out}} \cup S_{\text{in}}| \geq k$. Also, notice that, for every $x, y \in L_z$, there is an (x,y)-path $P^\star$ in $W^{(z)} \setminus (V(L_z) \setminus \{x, y\})$ that contains S' and avoids v. It is easy to see that there exist two disjoint paths Q_1, Q_2 in $\mathsf{compass}(W_0'^{(i\cdot b)})$ connecting $\{s_{\hat{P}}, t_{\hat{P}}\}$ with $\{x, y\}$ and that these paths can be picked to be internally disjoint from $\hat{P}$ and $P^\star$. Thus, if $\tilde{P}''$ is the graph obtained from $\tilde{P}'$ after removing all internal vertices of $\hat{P}$, then $\tilde{P}'' \cup Q_1 \cup Q_2 \cup P^\star$ is the claimed (s,t)-path that contains $S' \cup S_{\text{out}}$ and avoids v (see Figure 3). ◀

2.3 Proof of Theorem 1

Let (G, M) be a framework, where G is a planar graph and M is a linear matroid given by its representation over a finite filed or the field of rationals, and let $k \in \mathbb{N}$. We set $q = g(k)$, where g is the function of Lemma 6. Keep in mind that $g(k) = 2^{\mathcal{O}(k \log k)}$. We describe an algorithm $\mathcal{A}$ that solves Maximum Rank (s,t)-Path.

Our algorithm $\mathcal{A}$ first calls the single-exponential time 2-approximation algorithm for treewidth of Korhonen [33] for G and q which runs in time $2^q \cdot n = 2^{2^{\mathcal{O}(k \log k)}} \cdot n$ and outputs either a tree decomposition of G of width at most $2q$ or a report that the treewidth of G is larger

than q. In the first possible output, we can solve the problem using our dynamic programming algorithm which runs in time $2^{q^{\mathcal{O}(1)}} \cdot (|G| + \|M\|)^{\mathcal{O}(1)} = 2^{2^{\mathcal{O}(k \log k)}} \cdot (|G| + \|M\|)^{\mathcal{O}(1)}$. In the second possible output (i.e., where G has treewidth at least q), we apply the algorithm of Lemma 6 and, in time $2^{2^{\mathcal{O}(k \log k)}} \cdot (|G| + \|M\|)^{\mathcal{O}(1)}$, we either report a positive answer, or find a vertex $v \in V(G)$ such that (G, M, k, s, t) and $(G \setminus v, M \setminus v, k, s, t)$ are equivalent instances of the problem. If the latter happens, we recursively run $\mathcal{A}$ for the framework $(G \setminus v, M \setminus v)$. Observe that the overall running time of $\mathcal{A}$ is $2^{2^{\mathcal{O}(k \log k)}} \cdot (|G| + \|M\|)^{\mathcal{O}(1)}$.

3 Conclusion

In this paper, we provide a deterministic FPT algorithm for Maximum Rank (s,t)-Path for frameworks (G, M), where G is a planar graph and M is represented over a finite field or the rationals. Let us conclude by discussing some open research directions.

Since the algorithm of [15] for Maximum Rank (s,t)-Path runs in time $2^{\mathcal{O}(k^2 \log(k+q))} n^{\mathcal{O}(1)}$, a natural question is whether one can drop the double-exponential dependence on the parameter k on the running time of the algorithm of Theorem 1. The main bottleneck is the bound the treewidth of a graph that contains no irrelevant vertices. In particular, our approach to detect irrelevant vertices requires a recursive zooming into a given wall of the graph in order to find a packing of $k+1$-many k-walls with compasses of specific rank. To perform this zooming, one should ask for the initial wall to be of height at least $k^{\mathcal{O}(k)}$. It is unclear whether we can circumvent this argument and detect irrelevant vertices if the initial wall has height linear (or even polynomial) in k.

As mentioned in the introduction, the method of [15] gives a randomized algorithm for the more general problem of Maximum Rank (S,T)-Linkage. In this paper, we focus on the special case where $|S| = |T| = 1$ and one could ask whether our techniques can be applied to solve the general problem of detecting (S,T)-linkages of large rank for frameworks with planar graphs and matroids represented over finite fields. Such a generalization of our results does not seem to be trivial and therefore we leave this as an open research direction.

Another natural question to ask is whether our approach can be generalized to obtain deterministic FPT algorithms for frameworks with more general classes of graphs. While it seems plausible to extend the applicability of the irrelevant vertex technique arguments up to graphs that exclude a graph as a minor, such a proof would be highly technical. For frameworks with general graphs, it is very unclear whether one can achieve rerouting that does not decrease the rank and therefore allow an irrelevant vertex argument to go through.

Also, in the lines of [15], an interesting open question is whether we can obtain a deterministic FPT algorithm for Maximum Rank (s,t)-Path for frameworks with matroids not representable in finite fields of small order or in the field of rationals. For example, uniform matroids, and more generally transversal matroids, are representable over a finite field, but the field of representation must be large enough. While the approach of [15] also gives a *randomized* FPT algorithm for frameworks of transversal matroids, our dynamic programming subroutine relies on the efficient computation of representative sets, which requires a linear representation of the input matroid. We stress that this is the only place in the proof of Theorem 1 requiring a linear representation of the matroid. Another interesting open question, is whether Maximum Rank (s,t)-Path is FPT when parameterized by k and the treewidth if the input matroid is given by its independence oracle.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

2 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. A complexity dichotomy for hitting connected minors on bounded treewidth graphs: the chair and the banner draw the boundary. In *Proc. of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 951–970, 2020. doi:10.1137/1.9781611975994.57.

3 Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014. doi:10.1137/110839229.

4 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.*, 87:119–139, 2017. doi:10.1016/j.jcss.2017.03.003.

5 Andreas Björklund, Thore Husfeldt, and Nina Taslaman. Shortest cycle through specified elements. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1747–1753. SIAM, 2012. doi:10.1137/1.9781611973099.139.

6 Hajo Broersma, Xueliang Li, Gerhard J Woeginger, and Shenggui Zhang. Paths and cycles in colored graphs. *Australas. J Comb.*, 31:299–312, 2005.

7 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

8 Chandra Chekuri and Martin Pál. A recursive greedy algorithm for walks in directed graphs. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 245–253. IEEE Computer Society, 2005.

9 Johanne Cohen, Giuseppe F. Italiano, Yannis Manoussakis, Nguyen Kim Thang, and Hong Phong Pham. Tropical paths in vertex-colored graphs. *J. Comb. Optim.*, 42(3):476–498, 2021. doi:10.1007/s10878-019-00416-y.

10 Basile Couëtoux, Elie Nakache, and Yann Vaxès. The maximum labeled path problem. *Algorithmica*, 78(1):298–318, 2017. doi:10.1007/s00453-016-0155-6.

11 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

12 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.

13 Eduard Eiben, Tomohiro Koana, and Magnus Wahlström. Determinantal sieving. *CoRR*, abs/2304.02091, 2023. doi:10.48550/arXiv.2304.02091.

14 Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, Daniel Lokshtanov, and Giannos Stamoulis. Shortest cycles with monotone submodular costs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2214–2227. SIAM, 2023. doi:10.1137/1.9781611977554.ch83.

15 Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, Kirill Simonov, and Giannos Stamoulis. Fixed-parameter tractability of maximum colored path and beyond. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3700–3712. SIAM, 2023. doi:10.1137/1.9781611977554.ch142.

16 Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, and Giannos Stamoulis. Computing paths of large rank in planar frameworks deterministically. *CoRR*, abs/2305.01993, 2023. doi:10.48550/arXiv.2305.01993.

17 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Modification to Planarity is Fixed Parameter Tractable. In *Proc. of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2019.28.

18 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

19 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Hitting topological minors is FPT. In *Proc. of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1317–1326. ACM, 2020. doi:10.1145/3357713.3384318.

20 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (connected) dominating set on H-minor-free graphs. In *Proc. of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 82–93, 2012. doi:10.1137/1.9781611973099.7.

21 Petr A. Golovach, Marcin Kaminski, Spyridon Maniatis, and Dimitrios M. Thilikos. The parameterized complexity of graph cyclability. *SIAM J. Discret. Math.*, 31(1):511–541, 2017.

22 Petr A. Golovach, Giannos Stamoulis, and Dimitrios M. Thilikos. Model-checking for first-order logic with disjoint paths predicates in proper minor-closed graph classes. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3684–3699. SIAM, 2023. doi:10.1137/1.9781611977554.ch141.

23 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proc. of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 479–488. ACM, 2011. doi:10.1145/1993636.1993700.

24 Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Włodarczyk. Vertex deletion parameterized by elimination distance and even less. In *Proc. of the 53rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 1757–1769, 2021. doi:10.1145/3406325.3451068.

25 Ken-ichi Kawarabayashi. An improved algorithm for finding cycles through elements. In *13th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 5035 of *Lecture Notes in Computer Science*, pages 374–384. Springer, 2008. doi:10.1007/978-3-540-68891-4_26.

26 Ken-ichi Kawarabayashi. Planarity allowing few error vertices in linear time. In *Proc. of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 639–648, 2009. doi:10.1109/FOCS.2009.45.

27 Ken-ichi Kawarabayashi, Stephan Kreutzer, and Bojan Mohar. Linkless and flat embeddings in 3-space and the unknot problem. In *Proc. of the 2010 Annual Symposium on Computational Geometry (SoCG)*, pages 97–106. ACM, 2010. doi:10.1145/1810959.1810975.

28 Ken-ichi Kawarabayashi, Bojan Mohar, and Bruce A. Reed. A simpler linear time algorithm for embedding graphs into an arbitrary surface and the genus of graphs of bounded tree-width. In *Proc. of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 771–780, 2008. doi:10.1109/FOCS.2008.53.

29 Ken-ichi Kawarabayashi and Bruce A. Reed. Hadwiger's conjecture is decidable. In *Proc. of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 445–454, 2009. doi:10.1145/1536414.1536476.

30 Ken-ichi Kawarabayashi and Bruce A. Reed. Odd cycle packing. In *Proc. of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 695–704, 2010. doi:10.1145/1806689.1806785.

31 Jon M. Kleinberg. Decision algorithms for unsplittable flow and the half-disjoint paths problem. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 530–539. ACM, 1998.

32 Yusuke Kobayashi and Ken-ichi Kawarabayashi. Algorithms for finding an induced cycle in planar graphs and bounded genus graphs. In *Proc. of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1146–1155. ACM-SIAM, 2009. URL: https://dl.acm.org/doi/10.5555/1496770.1496894.

33 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 184–192. IEEE, 2021. doi:10.1109/FOCS52979.2021.00026.

34 Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming – Volume Part I*, ICALP '08, pages 575–586, Berlin, Heidelberg, 2008. Springer-Verlag. doi:10.1007/978-3-540-70575-8_47.

35 Ioannis Koutis and Ryan Williams. Algebraic fingerprints for faster algorithms. *Commun. ACM*, 59(1):98–105, December 2015. doi:10.1145/2742544.

36 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Trans. Algorithms*, 14(2):14:1–14:20, 2018. doi:10.1145/3170444.

37 L. Lovász. Flats in matroids and geometric graphs. In *Combinatorial surveys (Proc. Sixth British Combinatorial Conf., Royal Holloway Coll., Egham, 1977)*, pages 45–86, 1977.

38 L. Lovász and M. D. Plummer. *Matching theory*, volume 121 of *North-Holland Mathematics Studies*. North-Holland Publishing Co., Amsterdam; North-Holland Publishing Co., Amsterdam, 1986. Annals of Discrete Mathematics, 29.

39 László Lovász. *Graphs and geometry*, volume 65 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, Providence, RI, 2019. doi:10.1090/coll/065.

40 Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012. doi:10.1007/s00453-010-9484-z.

41 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. doi:10.1007/BF01588971.

42 Neil Robertson and Paul D. Seymour. Graph minors XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

43 Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a planar graph. *J. Comb. Theory, Ser. B*, 62(2):323–348, 1994. doi:10.1006/jctb.1994.1073.

44 Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. k-apices of minor-closed graph classes. II. Parameterized algorithms. *ACM Transactions on Algorithms*, 18(3), 2022. doi:10.1145/3519028.

45 Magnus Wahlström. Abusing the Tutte matrix: An algebraic instance compression for the K-set-cycle problem. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 – March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 341–352. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.341.

Pattern-Avoiding Binary Trees – Generation, Counting, and Bijections

Petr Gregor
Department of Theoretical Computer Science and Mathematical Logic,
Charles University, Prague, Czech Republic

Torsten Mütze
Department of Computer Science, University of Warwick, United Kingdom
Department of Theoretical Computer Science and Mathematical Logic,
Charles University, Prague, Czech Republic

Namrata
Department of Computer Science, University of Warwick, Coventry, UK

Abstract

In this paper we propose a notion of pattern avoidance in binary trees that generalizes the avoidance of contiguous tree patterns studied by Rowland and non-contiguous tree patterns studied by Dairyko, Pudwell, Tyner, and Wynn. Specifically, we propose algorithms for generating different classes of binary trees that are characterized by avoiding one or more of these generalized patterns. This is achieved by applying the recent Hartung–Hoang–Mütze–Williams generation framework, by encoding binary trees via permutations. In particular, we establish a one-to-one correspondence between tree patterns and certain mesh permutation patterns. We also conduct a systematic investigation of all tree patterns on at most 5 vertices, and we establish bijections between pattern-avoiding binary trees and other combinatorial objects, in particular pattern-avoiding lattice paths and set partitions.

2012 ACM Subject Classification Mathematics of computing → Enumeration; Mathematics of computing → Permutations and combinations; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Generation, binary tree, pattern avoidance, permutation, bijection

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.33

Related Version *Full Version*: `https://arxiv.org/abs/2306.08420` [20]

Funding This work was supported by Czech Science Foundation grant GA 22-15272S.

1 Introduction

Pattern avoidance is a central theme in combinatorics and discrete mathematics. For example, in Ramsey theory one investigates how order arises in large unordered structures such as graphs, hypergraphs, or subsets of the integers. The concept also arises naturally in algorithmic applications. For example, Knuth [28] showed that the integer sequences that are sortable by one pass through a stack are precisely 231-avoiding permutations. Pattern-avoiding permutations are a particularly important and heavily studied strand of research, one that comes with its own associated conference "Permutation Patterns", held annually since 2003. While it may seem that pattern-avoiding permutations are somewhat limited in scope, via suitable bijections they actually encode many objects studied in other branches of combinatorics. Pattern avoidance has also been studied directly in these other classes of objects, such as trees [38, 13, 12, 11, 15, 37, 6, 1, 16], set partitions [29, 24, 25, 26, 18, 22, 32, 33, 39, 19, 23, 17, 8], lattice paths [40, 5, 2, 4], heaps [30], matchings [7], and rectangulations [34]. In this work, we focus on binary trees, a class of objects that is fundamental within computer science, and also a classical Catalan family.

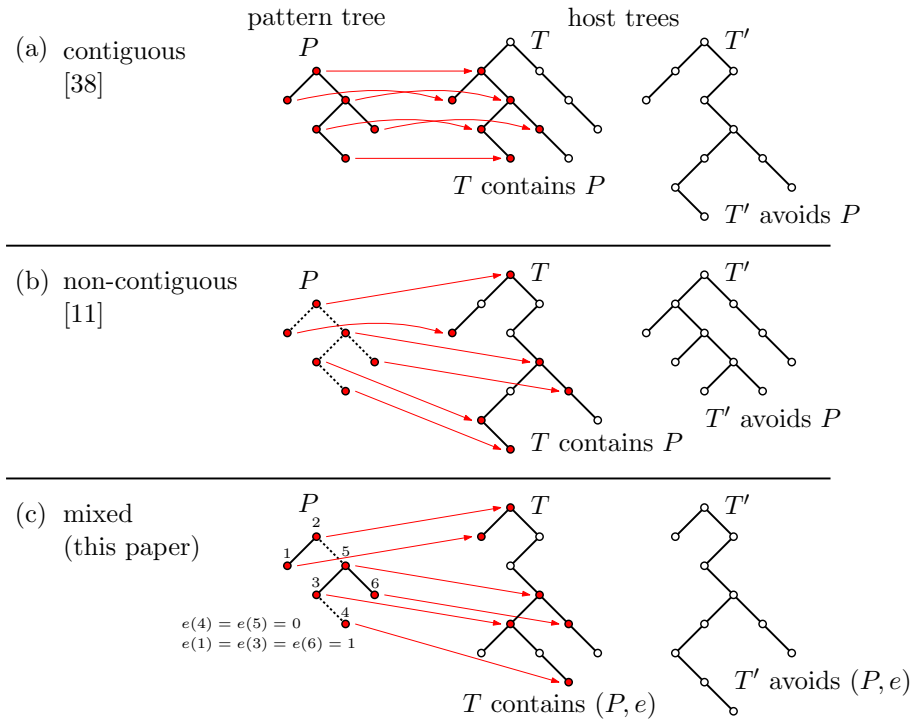


Figure 1 Illustration of different notions of pattern containment in binary trees. Contiguous edges are drawn solid, whereas non-contiguous edges are drawn dotted.

So far, two different notions of pattern avoidance in binary trees have been studied in the literature. We consider a binary tree T, which serves as the host tree, and another binary tree P, which serves as the pattern tree. Rowland [38] considered a *contiguous* notion of pattern containment, where T contains P if P is present as an induced subtree of T; see Figure 1 (a). He devised an algorithm to compute the generating function for the number of n-vertex binary trees that avoid P, and he showed that this generating function is always algebraic. Dairyko, Pudwell, Tyner, and Wynn [11] considered a *non-contiguous* notion of pattern containment, where T contains P if P is present as a "minor" of T; see Figure 1 (b). They discovered the remarkable phenomenon that for any two distinct k-vertex pattern trees P and P', the number of n-vertex host trees that avoid P is the same as the number of trees that avoid P', i.e., P and P' are *Wilf-equivalent* patterns. They also obtain the corresponding generating function (which is independent of P, but only depends on k and n).

In this paper, we consider *mixed* tree patterns, which generalize both of the two aforementioned types of tree patterns, by specifying separately for each edge of P whether it is considered contiguous or non-contiguous, i.e., whether its end vertices in the occurrence of the pattern must be in a parent-child or ancestor-descendant relationship (in the correct direction left/right), respectively; see Figure 1 (c). Observe that the notions of tree patterns considered in [38] and [11] are the tree analogues of consecutive [14] and classical permutation patterns, respectively. Our new notion of mixed patterns is the tree analogue of vincular permutation patterns [3], which generalize classical and consecutive permutation patterns.

1.1 The Lucas–Roelants van Baronaigien–Ruskey algorithm

One of the goals in this paper is to generate different classes of binary trees, i.e., we seek an algorithm that visits every tree from the class exactly once. Our starting point is a classical result due to Lucas, Roelants van Baronaigien, and Ruskey [31], which asserts that all n-vertex binary trees can be generated by tree rotations, i.e., every tree is obtained from its predecessor by a single *tree rotation* operation; see Figures 2 and 3.

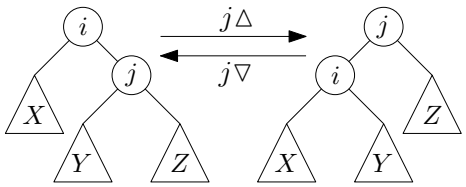


Figure 2 Rotation in binary trees.

The algorithm is an instance of a *combinatorial Gray code* [41, 35], which is a listing of objects such that any two consecutive objects differ in a "small local" change. The aforementioned Gray code algorithm for binary trees can be implemented in time $\mathcal{O}(1)$ per generated tree.

Williams [42] discovered a stunningly simple description of the Lucas–Roelants van Baronaigien–Ruskey Gray code for binary trees via the following greedy algorithm, which is based on labeling the vertices with $1, \ldots, n$ according to the search tree property: Start with the right path, and then repeatedly perform a tree rotation with the largest possible vertex that creates a previously unvisited tree.

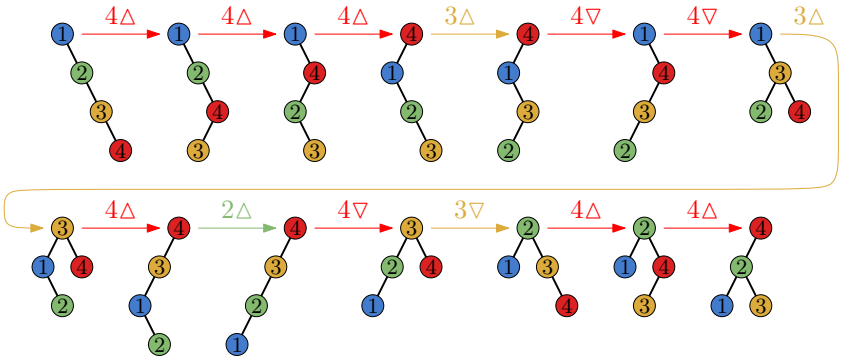


Figure 3 The Lucas–Roelants van Baronaigien–Ruskey algorithm to generate all binary trees with $n = 4$ vertices by tree rotations. The vertices are labeled with $1, 2, 3, 4$ according to the search tree property.

1.2 Our results

It is well known that binary trees are in bijection with 231-avoiding permutations. Our first result generalizes this bijection, by establishing a one-to-one correspondence between mixed binary tree patterns and mesh permutation patterns, a generalization of classical permutation patterns due to Brändén and Claesson [9]. Specifically, we show that n-vertex binary trees that avoid a particular (mixed) tree pattern P are in bijection with 231-avoiding permutations that avoid a corresponding mesh pattern $\sigma(P)$ (see Theorem 2 below).

This bijection enables us to apply the Hartung–Hoang–Mütze–Williams generation framework [21], which is based on permutations. We thus obtain algorithms for efficiently generating different classes of pattern-avoiding binary trees, which work under some mild conditions on the tree pattern(s). These algorithms are all based on a simple greedy algorithm, which generalizes Williams' algorithm for the Lucas–Roelants van Baronaigien–Ruskey Gray code of binary trees (see Algorithm S, Algorithm H, and Theorems 3 and 4, respectively). Specifically, instead of tree rotations our algorithms use a more general operation that we refer to as a *slide*. We implemented our generation algorithm in C++, and we made it available for download and experimentation on the Combinatorial Object Server [10].

For our new notion of mixed tree patterns, we conduct a systematic investigation of all tree patterns on up to 5 vertices. This gives rise to many counting sequences, some already present in the OEIS [36] and some new to it, giving rise to several interesting conjectures. In this work we establish most of these as theorems, by proving bijections between different classes of pattern-avoiding binary trees and other combinatorial objects, in particular pattern-avoiding lattice paths (Section 6 and [20, Sec. 7.2.4]) and set partitions ([20, Thm. 15]).

1.3 Outline of this paper

In Section 2 we introduce notations that will be used throughout the paper. In Section 3 we establish a bijection between binary trees patterns and mesh patterns. In Section 4 we present our algorithms for generating pattern-avoiding binary trees. In Section 5 we report on our computational results for all small tree patterns. In Section 6 we prove bijections between pattern-avoiding binary trees and Motzkin paths. Some open problems are discussed in Section 7. Due to space constraints, this extended abstract omits proofs, and several further results, illustrations and tables; see [20].

2 Preliminaries

In this section we introduce a few general definitions related to binary trees, and we define our notion of pattern avoidance for those objects.

2.1 Binary tree notions

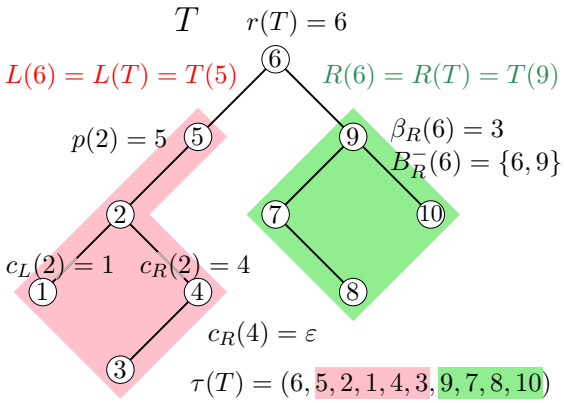


Figure 4 Definitions related to binary trees.

We consider binary trees whose vertex set is a set of consecutive integers $\{i, i+1, \ldots, j\}$. In particular, we write $\mathcal{T}_n$ for the set of binary trees with the vertex set $[n] := \{1, 2, \ldots, n\}$. The vertex labels of each tree are defined uniquely by the *search tree property*, i.e., for any vertex i, all its left descendants are smaller than i and all its right descendants are greater than i. The special empty tree with $n = 0$ vertices is denoted by ε, so $\mathcal{T}_0 = \{\varepsilon\}$. The following definitions are illustrated in Figure 4. For any binary tree T, we denote the root of T by $r(T)$. For any vertex i of T, its left and right child are denoted by $c_L(i)$ and $c_R(i)$, respectively, and its parent is denoted by $p(i)$. If i does not have a left child, a right child or a parent, then we define $c_L(i) := \varepsilon$, $c_R(i) := \varepsilon$, or $p(i) := \varepsilon$, respectively. Furthermore, we write $T(i)$ for the subtree of T rooted at i. Also, we define $L(i) := T(c_L(i))$ if $c_L(i) \neq \varepsilon$ and $L(i) := \varepsilon$ otherwise, and $R(i) := T(c_R(i))$ if $c_R(i) \neq \varepsilon$ and $R(i) := \varepsilon$ otherwise. The subtrees rooted at the left and right child of the root are denoted by $L(T)$ and $R(T)$, respectively, i.e., we have $L(T) = L(r(T))$, and similarly $R(T) = R(r(T))$. A *left path* is a binary tree in which no vertex has a right child. A *left branch* in a binary tree is a subtree that is isomorphic to a left path. The notions *right path* and *right branch* are defined analogously, by interchanging left and right.

We associate $T \in \mathcal{T}_n$ with a permutation $\tau(T)$ of $[n]$ defined by

$$\tau(T) := \big(r(T), \tau(L(T)), \tau(R(T))\big), \tag{1}$$

where the base case of the empty tree ε is defined to be the empty permutation $\tau(\varepsilon) := \varepsilon$. In words, $\tau(T)$ is the sequence of vertex labels obtained from a preorder traversal of T, i.e., we first record the label of the root and then recursively record labels of its left subtree followed by labels of its right subtree. Note that the right path $T \in \mathcal{T}_n$ satisfies $\tau(T) = \mathrm{id}_n$, the identity permutation.

For any vertex i we let $\beta_R(i)$ denote the number of vertices on the right branch starting at i, with the special case $\beta_R(\varepsilon) := 0$. We also define $B_R^-(i) := \{c_R^{j-1}(i) \mid j = 1, \ldots, \beta_R(i) - 1\}$ as the corresponding sets of vertices on this branch except the last one.

2.2 Pattern-avoiding binary trees

Our notion of pattern avoidance in binary trees generalizes the two distinct notions considered in [38] and [11] (recall Figure 1). This definition is illustrated in Figure 5. A *tree pattern* is a pair (P, e) where $P \in \mathcal{T}_k$ and $e\colon [k] \setminus r(P) \to \{0, 1\}$. For any vertex $i \in [k] \setminus r(P)$, a value $e(i) = 0$ is interpreted as the edge leading from i to its parent $p(i)$ being non-contiguous, whereas a value $e(i) = 1$ is interpreted as this edge being contiguous. In our figures, edges $(i, p(i))$ in P with $e(i) = 1$ are drawn solid, and edges with $e(i) = 0$ are drawn dotted. Formally, a tree $T \in \mathcal{T}_n$ *contains* the pattern (P, e) if there is an injective mapping $f\colon [k] \to [n]$ satisfying the following conditions:

- **(i)** For every edge $(i, p(i))$ of P with $e(i) = 1$, we have that $f(i)$ is a child of $f(p(i))$ in T. Specifically, if $i = c_L(p(i))$ then $f(i)$ is the left child of $f(p(i))$, i.e., we have $f(i) = c_L(f(p(i)))$, whereas if $i = c_R(p(i))$ then $f(i)$ is the right child of $f(p(i))$, i.e., we have $f(i) = c_R(f(p(i)))$.
- **(ii)** For every edge $(i, p(i))$ of P with $e(i) = 0$, we have that $f(i)$ is a descendant of $f(p(i))$ in T. Specifically, if $i = c_L(p(i))$, then $f(i)$ is a left descendant of $f(p(i))$, i.e., we have $f(i) \in L(f(p(i)))$, whereas if $i = c_R(p(i))$, then $f(i)$ is a right descendant of $f(p(i))$, i.e., we have $f(i) \in R(f(p(i)))$.

We can retrieve the notions of contiguous and non-contiguous pattern containment used in [38] and [11] as special cases by defining $e(i) := 1$ for all $i \in [k] \setminus r(P)$, or $e(i) := 0$ for all $i \in [k] \setminus r(P)$, respectively.

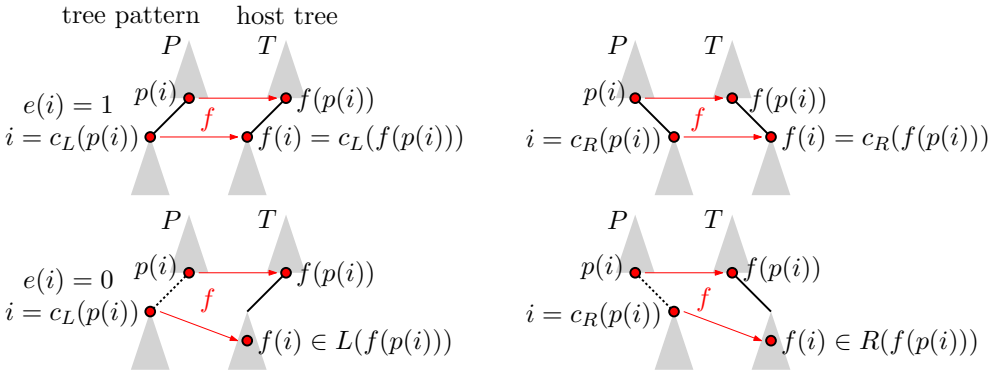


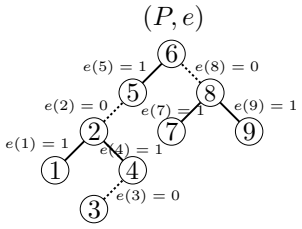
Figure 5 Illustration of our notion of pattern containment in binary trees.

If T does not contain (P, e), then we say that T *avoids* (P, e). Furthermore, we define the set of binary trees with n vertices that avoid the pattern (P, e) as

$$\mathcal{T}_n(P, e) := \{T \in \mathcal{T}_n \mid T \text{ avoids } (P, e)\}.$$

For avoiding multiple patterns $(P_1, e_1), \ldots, (P_\ell, e_\ell)$ simultaneously, we define

$$\mathcal{T}_n\big((P_1, e_1), \ldots, (P_\ell, e_\ell)\big) := \bigcap_{i=1}^{\ell} \mathcal{T}_n(P_i, e_i).$$



$\tau(P) = 652143879$
$e = (e(5), e(2), e(1), e(4), e(3),$
$\quad e(8), e(7), e(9)) = 10110011$
$(P, e) = (652143879, 10110011)$

Figure 6 Compact encoding of binary tree patterns.

We often write a tree pattern (P, e), $P \in \mathcal{T}_k$, in compact form as a pair $\big(\tau(P), (e(\tau_2), \ldots, e(\tau_k))\big)$ where $\tau(P) = (\tau_1, \tau_2, \ldots, \tau_k)$; see Figure 6. In words, the tree P is specified by the preorder permutation $\tau(P)$, and the function e is specified by the sequence of values for all vertices except the root in the preorder sequence, i.e., this sequence has length $k - 1$.

For any tree pattern (P, e), we write $\mu(P, e)$ for the pattern obtained by mirroring the tree, i.e., by changing left and right. Note that the mirroring operation changes the vertex labels so that the search tree property is maintained, specifically the vertex i becomes $n+1-i$. Trivially, we have $\mathcal{T}_n(\mu(P, e)) = \mu(\mathcal{T}_n(P, e))$, in particular (P, e) and $\mu(P, e)$ are Wilf-equivalent.

3 Encoding binary trees by permutations

In this section we establish that avoiding a tree pattern in binary trees is equivalent to avoiding a corresponding mesh pattern in 231-avoiding permutations (Theorem 2 below).

3.1 Pattern-avoiding permutations

We write S_n for the set of all permutations of $[n]$. Given two permutations $\pi \in S_n$ and $\tau \in S_k$, we say that π *contains* τ *as a pattern* if there is a sequence of indices $\nu_1 < \cdots < \nu_k$, such that $\pi(\nu_1), \ldots, \pi(\nu_k)$ are in the same relative order as $\tau = \tau(1), \ldots, \tau(k)$. If π does not contain τ, then we say that π *avoids* τ. We write $S_n(\tau)$ for the permutations from S_n that avoid the pattern τ. More generally, for multiple patterns $\tau_1, \ldots, \tau_\ell$ we define $S_n(\tau_1, \ldots, \tau_\ell) := \bigcap_{i=1}^{\ell} S_n(\tau_i)$, i.e., this is the set of permutations of length n that avoid each of the patterns $\tau_1, \ldots, \tau_\ell$. It is well known that preorder traversals of binary trees are in bijection with 231-avoiding permutations (see, e.g. [27]).

▶ **Lemma 1.** *The mapping* $\tau\colon \mathcal{T}_n \to S_n(231)$ *defined in* (1) *is a bijection.*

3.2 Mesh patterns

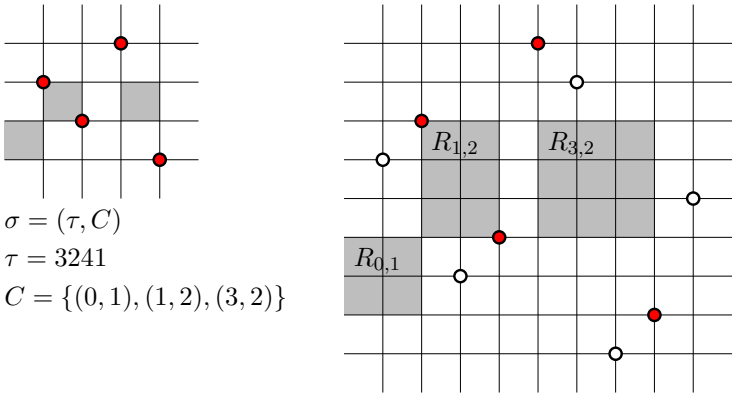


Figure 7 Illustration of mesh pattern containment.

Mesh patterns were introduced by Brändén and Claesson [9], and they generalize classical permutation patterns discussed in the previous section. We recap the required definitions; see Figure 7. The *grid representation* of a permutation $\pi \in S_n$ is defined as $G(\pi) := \{(i, \pi(i)) \mid i \in [n]\}$. Graphically, this is the permutation matrix corresponding to π.

A *mesh pattern* is a pair $\sigma := (\tau, C)$, where $\tau \in S_k$ and $C \subseteq \{0, \ldots, k\} \times \{0, \ldots, k\}$. In our figures, we depict σ by the grid representation of τ, and we shade all unit squares $[i, i+1] \times [j, j+1]$ for which $(i, j) \in C$. A permutation $\pi \in S_n$ *contains* the mesh pattern $\sigma = (\tau, C)$, if there is a sequence of indices $\nu_1 < \cdots < \nu_k$ such that the following two conditions hold:

- **(i)** The entries of $\pi(\nu_1), \ldots, \pi(\nu_k)$ are in the same relative order as $\tau = \tau(1), \ldots, \tau(k)$.
- **(ii)** We let $\lambda_1 < \cdots < \lambda_k$ be the values $\pi(\nu_1), \ldots, \pi(\nu_k)$ sorted in increasing order. For all pairs $(i, j) \in C$, we require that $G(\pi) \cap R_{i,j} = \emptyset$, where $R_{i,j}$ is the rectangular open set defined as $R_{i,j} := (\nu_i, \nu_{i+1}) \times (\lambda_j, \lambda_{j+1})$, using the sentinel values $\nu_0 := \lambda_0 := 0$ and $\nu_{k+1} = \lambda_{k+1} := n + 1$.

The first condition requires a match of the classical pattern τ in π. The second condition requires that $G(\pi)$ has no point in any of the regions $R_{i,j}$ that correspond to the shaded cells C of the pattern. Thus, the classical pattern $\tau \in S_k$ is the mesh pattern $(\tau, \emptyset)$.

3.3 From binary tree patterns to mesh patterns

In the following, for a given tree pattern (P, e), $P \in \mathcal{T}_k$, we construct a permutation mesh pattern $\sigma(P, e) = (\tau(P), C)$, consisting of the permutation $\tau(P)$ obtained by a preorder traversal of the tree P and a set of shaded cells C. These definitions are illustrated in Figures 8 and 9. We consider the inverse permutation of $\tau(P) \in S_k$, which we abbreviate to $\rho := \tau(P)^{-1} \in S_k$. The permutation ρ gives the position of each vertex in the preorder traversal $\tau(P)$ of P. Recall the definition of the set $B_R^-(i)$ given in Section 2.1. For any vertex $i \in [k]$ we define

$$C_i := \big\{(\rho(i) - 1, j) \mid j \in B_R^-(i)\big\}, \tag{2a}$$

and for any $i \in [k] \setminus r(P)$ we define

$$C'_i := \begin{cases} \emptyset & \text{if } e(i) = 0, \\ \big\{\big(\rho(i) - 1, \min P(i) - 1\big),\ \big(\rho(i) - 1, \max P(i)\big)\big\} & \text{if } e(i) = 1. \end{cases} \tag{2b}$$

Then the mesh pattern $\sigma(P, e)$ corresponding to the tree pattern (P, e) is defined as

$$\sigma(P, e) := \Big(\tau(P), \bigcup\nolimits_{i \in [k]} C_i \cup \bigcup\nolimits_{i \in [k] \setminus r(P)} C'_i\Big). \tag{2c}$$

In words, for every pair of vertices (not necessarily distinct and not necessarily forming an edge) except the last vertex on a maximal right branch we shade the cell directly left of the smaller vertex and directly above the larger vertex, and for every edge $(i, p(i))$ with $e(i) = 1$ we shade two additional cells to the left and bottom/top of the submatrix corresponding to the subtree $P(i)$.

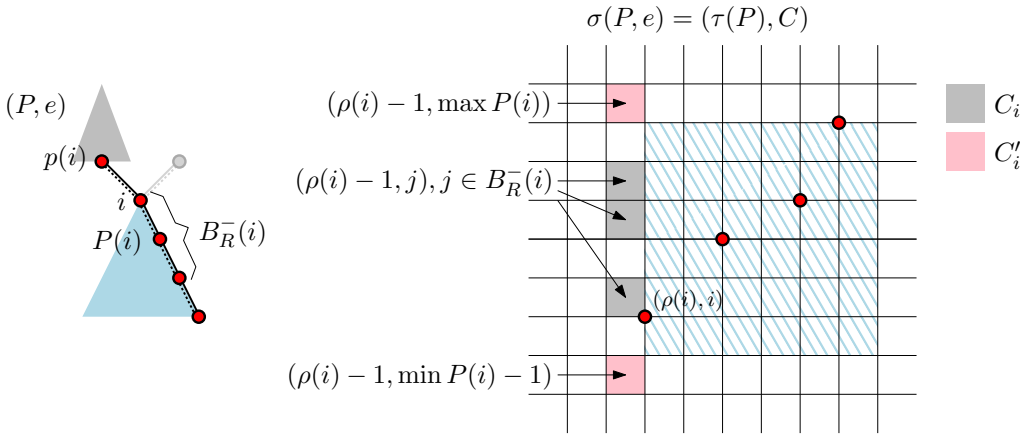


Figure 8 Schematic illustration of the definition of the mesh pattern $\sigma(P, e)$ for a tree pattern (P, e). The edges of the tree P can be contiguous or non-contiguous, and are therefore drawn half solid and half dotted. In the tree shown in the figure, i is the right child of $p(i)$, but it might also be the left child of $p(i)$ (faint lines). On the right, the shaded cells belong to the mesh pattern, and the hatched region corresponds to the submatrix given by the subtree $P(i)$.

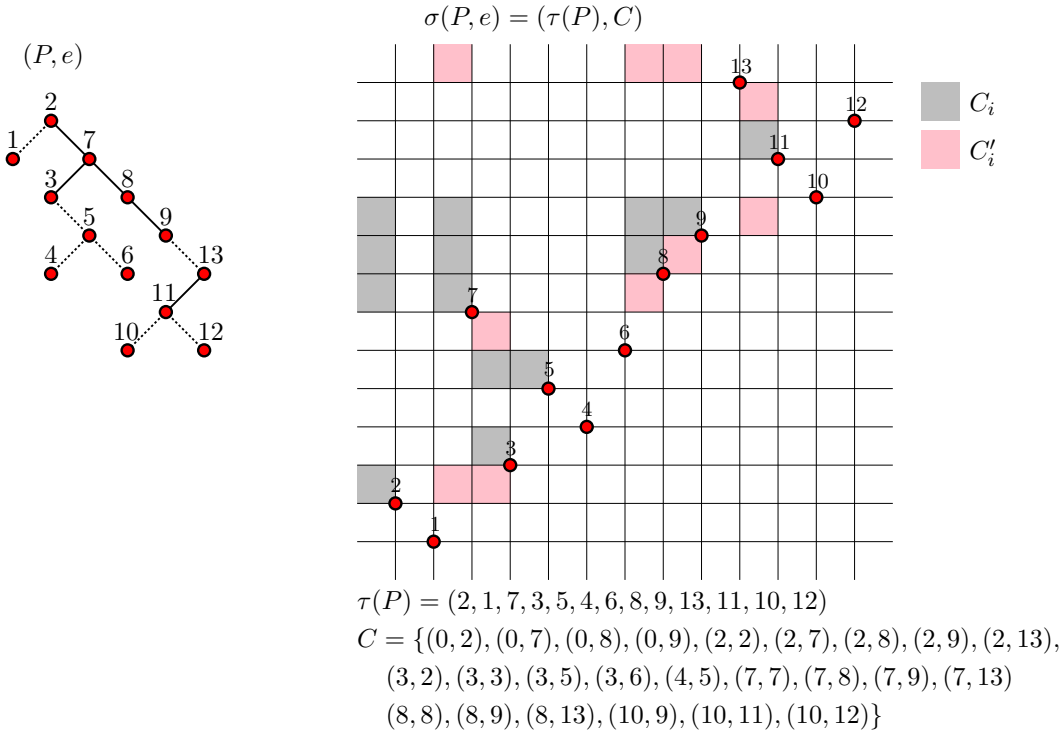


Figure 9 Specific example of the mesh pattern $\sigma(P, e)$ corresponding to a tree pattern (P, e).

The following generalization of Lemma 1 is the main result of this section. Our theorem also generalizes Theorem 12 from [37], which is obtained as the special case when all edges of P are non-contiguous, i.e., $e(i) = 0$ for all $i \in [k] \setminus r(P)$.

▶ **Theorem 2.** *For any tree pattern (P, e), $P \in \mathcal{T}_k$, consider the mesh pattern $\sigma(P, e) = (\tau(P), C)$ defined in (2). Then the mapping $\tau \colon \mathcal{T}_n(P, e) \to S_n(231, \sigma(P, e))$ is a bijection.*

This theorem extends naturally to avoiding multiple tree patterns $(P_1, e_1), \ldots, (P_\ell, e_\ell)$, i.e., $\tau \colon \mathcal{T}_n((P_1, e_1), \ldots, (P_\ell, e_\ell)) \to S_n(231, \sigma(P_1, e_1), \ldots, \sigma(P_\ell, e_\ell))$ is a bijection. The proof of Theorem 2 can be found in [20].

4 Generating pattern-avoiding binary trees

In this section we apply the Hartung–Hoang–Mütze–Williams generation framework to pattern-avoiding binary trees. The main results are simple and efficient algorithms (Algorithm S and Algorithm H) to generate different classes of pattern-avoiding binary trees, subject to some mild constraints on the tree pattern(s) that are inherited from applying the framework (Theorems 3 and 4, respectively).

4.1 Tree rotations and slides

A natural and well-studied operation on binary trees are tree rotations; see Figure 2. We consider a tree $T \in \mathcal{T}_n$ and one of its edges (i, j) with $j = c_R(i)$, and we let Y be the left subtree of j, i.e., $Y := L(j)$. A *rotation of the edge (i, j)* yields the tree obtained by the following modifications: The child i of $p(i)$ is replaced by j (unless $p(i) = \varepsilon$ in T), i becomes

the left child of j, and Y becomes the right subtree of i. We denote this operation by $j\triangle$, and we refer to it as *up-rotation of j*, indicating that the vertex j moves up. The operation $j\triangle$ is well-defined if and only if j is not the root and $p(j) < j$, or equivalently $j = c_R(p(j))$. The inverse operation is denoted by $j\triangledown$, and we refer to it as *down-rotation of j*, indicating that the vertex j moves down. The operation $j\triangledown$ is well-defined if and only if j has a left child (which must be smaller), i.e., $c_L(j) \neq \varepsilon$. An *up-slide* or *down-slide of j by d steps* is a sequence of d up- or down-rotations of j, respectively, which we write as $(j\triangle)^d$ and $(j\triangledown)^d$.

4.2 A simple greedy algorithm

We use the following simple greedy algorithm to generate a set of binary trees $\mathcal{L}_n \subseteq \mathcal{T}_n$. We say that a slide is *minimal* (w.r.t. $\mathcal{L}_n$), if every slide of the same vertex in the same direction by fewer steps creates a binary tree that is not in $\mathcal{L}_n$.

Algorithm S *(Greedy slides).* This algorithm attempts to greedily generate a set of binary trees $\mathcal{L}_n \subseteq \mathcal{T}_n$ using minimal slides starting from an initial binary tree $T_0 \in \mathcal{L}_n$.

S1. [Initialize] Visit the initial tree T_0.

S2. [Slide] Generate an unvisited binary tree from $\mathcal{L}_n$ by performing a minimal slide of the largest possible vertex in the most recently visited binary tree. If no such slide exists, or the direction of the slide is ambiguous, then terminate. Otherwise visit this binary tree and repeat S2.

To illustrate the algorithm, consider the example in Figure 10. Suppose we choose the right path T_1 shown in the figure as initial tree for the algorithm, i.e., $T_0 := T_1$. In the first iteration, Algorithm S performs an up-slide of the vertex 4 by three steps to obtain T_2. This up-slide is minimal, as an up-slide of 4 in T_1 by one or two steps creates the forbidden tree pattern (P, e). Note that any tree created from T_2 by a down-slide of 4 either contains the forbidden pattern or has been visited before. Consequently, the algorithm applies an up-slide of 3 by two steps, yielding T_3. After five more slides, the algorithm terminates with T_8, and at this point it has visited all eight trees in $\mathcal{T}_4(P, e)$.

Now consider the example in Figure 11, where the algorithm terminates after having visited six different trees from $\mathcal{T}_4(P, e)$. However, the set $\mathcal{T}_4(P, e)$ contains two more trees that are not visited by the algorithm.

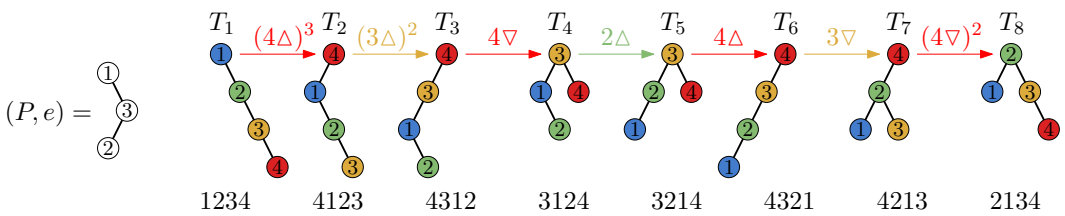


Figure 10 Run of Algorithm S that visits all binary trees in the set $\mathcal{T}_4(P, e)$. Below each tree T is the corresponding permutation $\tau(T)$.

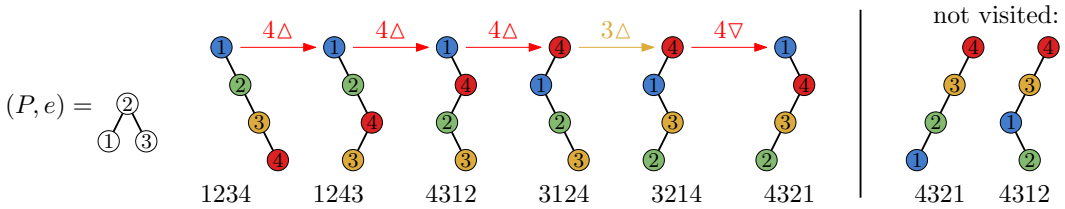


Figure 11 Run of Algorithm S that does not visit all binary trees in the set $\mathcal{T}_4(P, e)$.

We now formulate simple sufficient conditions on the tree pattern (P, e) ensuring that Algorithm S successfully visits all trees in $\mathcal{T}_n(P, e)$. Specifically, we say that a tree pattern (P, e), $P \in \mathcal{T}_k$, is *friendly*, if it satisfies the following three conditions; see Figure 12:

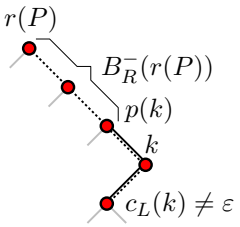


Figure 12 Definition of friendly tree patterns.

(i) We have $p(k) \neq \varepsilon$ and $c_L(k) \neq \varepsilon$, i.e., the largest vertex k is neither the root nor a leaf in P.

(ii) For every $j \in B_R^-(r(P)) \setminus r(P)$ we have $e(j) = 0$, i.e., the edges on the right branch starting at the root, except possibly the last one, are all non-contiguous.

(iii) If $e(k) = 1$, then we have $e(c_L(k)) = 0$, i.e., if the edge from k to its parent is contiguous, then the edge to its left child must be non-contiguous.

Note that for non-contiguous tree patterns, i.e., $e(i) = 0$ for all $i \in [k] \setminus r(P)$, conditions (ii) and (iii) are always satisfied. The following is our main result of this section.

▶ **Theorem 3.** *Let $(P_1, e_1), \ldots, (P_\ell, e_\ell)$ be friendly tree patterns. Then Algorithm S initialized with the tree $\tau^{-1}(\mathrm{id}_n)$ visits every binary tree from $\mathcal{T}_n((P_1, e_1), \ldots, (P_\ell, e_\ell))$ exactly once.*

Recall that $\tau^{-1}(\mathrm{id}_n)$ is the right path, i.e., the tree that corresponds to the identity permutation. Note that by condition (i) in the definition of friendly tree pattern, we have $\tau^{-1}(\mathrm{id}_n) \in \mathcal{T}_n((P_1, e_1), \ldots, (P_\ell, e_\ell))$. Theorem 3 can be proved by applying the Hartung–Hoang–Mütze–Williams generation framework [21]; see [20] for details. In particular, our notion of friendly tree patterns is inherited from the notion of tame mesh permutation patterns used in [21, Thm. 15].

4.3 Efficient implementation

We now describe an efficient implementation of Algorithm S. In particular, this implementation is *history-free*, i.e., it does not require to store all previously visited binary trees, but only maintains the current tree in memory. Algorithm H is a straightforward translation of the history-free Algorithm M presented in [34] from permutations to binary trees.

Algorithm H *(History-free minimal slides).* For friendly tree patterns $(P_1, e_1), \ldots, (P_\ell, e_\ell)$, this algorithm generates all binary trees from $\mathcal{T}_n$ that avoid $(P_1, e_1), \ldots, (P_\ell, e_\ell)$, i.e., the set $\mathcal{L}_n := \mathcal{T}_n((P_1, e_1), \ldots, (P_\ell, e_\ell)) \subseteq \mathcal{T}_n$ by minimal slides in the same order as Algorithm S. It maintains the current tree in the variable T, and auxiliary arrays $o = (o_1, \ldots, o_n)$ and $s = (s_1, \ldots, s_n)$.

H1. [Initialize] Set $T \leftarrow \tau^{-1}(\mathrm{id}_n)$, and $o_j \leftarrow \triangle$, $s_j \leftarrow j$ for $j = 1, \ldots, n$.

H2. [Visit] Visit the current binary tree T.

H3. [Select vertex] Set $j \leftarrow s_n$, and terminate if $j = 1$.

H4. [Slide] In the current binary tree T, perform a slide of the vertex j that is minimal w.r.t. $\mathcal{L}_n$, where the slide direction is up if $o_j = \triangle$ and down if $o_j = \triangledown$.

H5. [Update o and s] Set $s_n \leftarrow n$. If $o_j = \triangle$ and j is either the root or its parent is larger than j set $o_j = \triangledown$, or if $o_j = \triangledown$ and j has no left child set $o_j = \triangle$, and in both cases set $s_j \leftarrow s_{j-1}$ and $s_{j-1} = j - 1$. Go back to H2.

The two auxiliary arrays used by Algorithm H store the following information. The direction in which vertex j slides in the next step is maintained in the variable o_j. Furthermore, the array s is used to determine the vertex that slides in the next step. Specifically, the vertex j that slides in the next steps is retrieved from the last entry of the array s in step H3, by the instruction $j \leftarrow s_n$. The running time per iteration of the algorithm is governed by the time it takes to compute a minimal slide in step H4. This boils down to testing containment of the tree patterns (P_i, e_i), $i \in [\ell]$, in T.

▶ **Theorem 4.** *Let $(P_1, e_1), \ldots, (P_\ell, e_\ell)$ be friendly tree patterns with $P_i \in \mathcal{T}_{k_i}$ for $i \in [\ell]$. Then Algorithm H visits every binary tree from $\mathcal{T}_n((P_1, e_1), \ldots, (P_\ell, e_\ell))$ exactly once, in the same order as Algorithm S, in time $\mathcal{O}(n^2 \sum_{i=1}^{\ell} k_i^2)$ per binary tree.*

See [20] for a proof of Theorem 4.

5 Tree patterns on at most 5 vertices

We conducted systematic computer experiments with all tree patterns (P, e) on $k = 3, 4, 5$ vertices; see Tables 1, 2 and 3, respectively. Specifically, we computed the corresponding counting sequences $|\mathcal{T}_n(P, e)|$ for $n = 1, \ldots, 12$, and searched for matches within the OEIS [36]. There are three new counting sequences denoted by `NewA`, `NewB`, and `NewC`, which we added to the OEIS using the sequence numbers A365508, A365509, and A365510, respectively. All those counts were computed using Algorithm H for friendly tree patterns, and via brute-force methods for non-friendly tree patterns. As mirrored tree patterns are Wilf-equivalent, our tables only contain the lexicographically smaller of any such pair of mirrored trees, using the compact encoding described in Section 2.2.

It turns out that for some edges $(i, p(i))$ in a tree pattern (P, e), it is irrelevant whether the edge is considered contiguous ($e(i) = 1$) or non-contiguous ($e(i) = 0$). We have a theorem ([20, Thm. 11]) that describes these situations, and this theorem is used heavily in our tables, where those "don't care" values of e are denoted by the hyphen `-`. The statement and proof of this theorem are slightly technical, and so we omit it in this extended abstract.

Table 1 Tree patterns with 3 vertices. See Section 5 for explanations.

P	e	Friendly	Counts $\lvert\mathcal{T}_n(P,e)\rvert$ for $n=1,\ldots,12$													OEIS
123	0-		1	2	4	8	16	32	64	128	256	512	1024	2048	...	A000079
	1-		1	2	4	9	21	51	127	323	835	2188	5798	15511	...	A001006
132	--	0-, -0	1	2	4	8	16	32	64	128	256	512	1024	2048	...	A000079
213	--		1	2	4	8	16	32	64	128	256	512	1024	2048	...	A000079

Table 2 Tree patterns with 4 vertices.

P	e	Friendly	Counts $\lvert\mathcal{T}_n(P,e)\rvert$ for $n=1,\ldots,12$													OEIS
1234	00-		1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	01-		1	2	5	13	35	96	267	750	2123	6046	17303	49721	...	A005773
	10-		1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242
	11-		1	2	5	13	36	104	309	939	2905	9118	28964	92940	...	A036765
1243	0--	00-, 0-0	1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	1--		1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242
1324	0--		1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	1--		1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242
1423	0--, -0-	0--, -0-	1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	11-		1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242
1432	-0-	-0-	1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	-1-	01-	1	2	5	13	35	96	267	750	2123	6046	17303	49721	...	A005773
2134	-0-		1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	-1-		1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242
2143	-0-	-0-	1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	-1-	-10	1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242

Table 3 Tree patterns with 5 vertices.

P	e	Friendly	Counts $\lvert\mathcal{T}_n(P,e)\rvert$ for $n=1,\ldots,12$													OEIS
12345	000-		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	001-		1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	010-		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	011-		1	2	5	14	41	124	384	1210	3865	12482	40677	133572	...	A159772
	100-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	101-		1	2	5	14	41	124	383	1202	3819	12255	39651	129190	...	NewB→A365509
	110-		1	2	5	14	41	124	385	1221	3939	12886	42648	142544	...	A159768
	111-		1	2	5	14	41	125	393	1265	4147	13798	46476	158170	...	A036766
12354	00--	000-, 00-0	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	10--		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	11--		1	2	5	14	41	124	385	1221	3939	12886	42648	142544	...	A159768
12435	00--		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	10--		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	11--		1	2	5	14	41	124	385	1221	3939	12886	42648	142544	...	A159768
12534	00--, 0-0-	00--, 0-0-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	011-		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	10--, 1-0-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	111-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
12543	0-0-	0-0-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-	001-	1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	1-0-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	101-		1	2	5	14	41	124	383	1202	3819	12255	39651	129190	...	NewB→A365509
	111-		1	2	5	14	41	124	384	1211	3875	12548	41040	135370	...	A159770
13245	0-0-		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	1-0-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	1-1-		1	2	5	14	41	124	385	1221	3939	12886	42648	142544	...	A159768

P	e	Friendly	Counts $\lvert\mathcal{T}_n(P,e)\rvert$ for $n=1,\ldots,12$													OEIS
13254	0-0-	0-0-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-	0-10	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	1-0-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	1-1-		1	2	5	14	41	124	385	1220	3929	12822	42309	140922	...	A159771
14235	00--		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	10--		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	11--		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
14325	00--		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	10--		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	11--		1	2	5	14	41	124	384	1211	3875	12548	41040	135370	...	A159770
15234	0-0-, -00-	0-0-, -00-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-, -01-	0-1-, -01-	1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	110-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	111-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
15243	-0--, 0-0-	-0--, 0-0-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	011-	011-	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	110-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	111-		1	2	5	14	41	124	385	1220	3929	12822	42309	140922	...	A159771
15324	-0--	-0--	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--	01--	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	11--		1	2	5	14	41	124	384	1211	3875	12548	41040	135370	...	A159770
15423	-0--	-0--	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--	01--	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	-10-	010-	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	111-		1	2	5	14	41	124	384	1211	3875	12548	41040	135370	...	A159770
15432	-00-	-00-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-01-	-01-	1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	-10-	010-	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	-11-	011-	1	2	5	14	41	124	384	1210	3865	12482	40677	133572	...	A159772
21345	-00-		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-01-		1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	-10-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	-11-		1	2	5	14	41	124	385	1221	3939	12886	42648	142544	...	A159768
21354	-0--	-00-, -0-0	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-10-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	-11-		1	2	5	14	41	124	384	1212	3885	12613	41389	137055	...	A159773
21435	-0--		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-1--		1	2	5	14	41	124	385	1220	3929	12822	42309	140922	...	A159771
21534	-0--	-0--	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-10-	-10-	1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	-11-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
21543	-00-	-00-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-01-	-01-	1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	-10-	-10-	1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	-11-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
31245	0-0-		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	1-0-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	1-1-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
31254	0-0-	0-0-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-, 1-0-	0-10, 1-0-	1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	1-1-	1-10	1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
32145	0-0-		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-, 1-0-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	1-1-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769

6 Bijections between binary trees and Motzkin paths

In this section, we present bijections between pattern-avoiding binary trees and different types of Motzkin paths. For more bijections with other combinatorial objects, see [20]. Specifically, we consider lattice paths with steps $\mathtt{U} := (1,1)$, $\mathtt{D} := (1,-1)$, $\mathtt{F} := (1,0)$, and $\mathtt{D}_h := (1,-h)$ for $h \geq 2$. An *n-step Motzkin path* starts at $(0,0)$, ends at $(n,0)$, uses only steps `U`, `D` or `F`, and it never goes below the x-axis. We write $\mathcal{M}_n$ for the set of all n-step Motzkin paths (OEIS A001006). An *n-step Motzkin left factor* starts at $(0,0)$, uses n many steps `U`, `D` or `F`,

and it never goes below the x-axis. We write $\mathcal{L}_n$ for the set of all n-step Motzkin left factors (OEIS A005773). An *n-step Motzkin path with catastrophes* [4] starts at $(0,0)$, ends at $(n,0)$, uses only steps U, D, F, or $\mathtt{D}_h$ for $h \geq 2$, such that all $\mathtt{D}_h$-steps end on the x-axis, and it never goes below the x-axis (OEIS A054391). We write $\mathcal{C}_n$ for the set of all n-step Motzkin paths with catastrophes.

6.1 Bijection between $\mathcal{T}_n(123, 1\text{-})$ and Motzkin paths $\mathcal{M}_n$

This bijection is illustrated in Figure 13 (a). Consider a tree $T \in \mathcal{T}_n(P,e)$ where $(P,e) := (123, 1\text{-})$. Due to the forbidden pattern (P,e), every maximal right branch in T consists of one or two vertices, but not more. We map T to an n-step Motzkin path $f(T)$ as follows. Every maximal right branch in T consisting of one vertex i creates an F-step at position i in $f(T)$. Every maximal right branch in T consisting of two vertices i and j, where $j = c_R(i)$, creates a pair of U-step and D-step at the same height at positions i and j in $f(T)$, respectively. It is easy to verify that f is indeed a bijection between $\mathcal{T}_n(P,e)$ and $\mathcal{M}_n$.

Rowland [38] described a bijection between $\mathcal{T}_n(123, 1\text{-})$ and $\mathcal{M}_n$ that is different from f.

6.2 Bijection between $\mathcal{T}_n(1432, \text{-}1\text{-})$ and Motzkin left factors $\mathcal{L}_{n-1}$

This bijection is illustrated in Figure 13 (b), and it uses as a building block the bijection f defined in the previous section. Instead of $(1432, \text{-}1\text{-})$, we consider the mirrored tree pattern $(P,e) := \mu(1432, \text{-}1\text{-}) = (4123, \text{-}1\text{-})$ for convenience. Consider a tree $T \in \mathcal{T}_n(P,e)$. We define $b := \beta_R(r(T))$ and $r_i := c_R^{i-1}(r(T))$ for $i = 1, \ldots, b$, i.e., we consider the right branch $(r_1, \ldots, r_b)$ starting from the root of T. Due to the forbidden tree pattern (P,e), each subtree $L(r_i)$ for $i = 1, \ldots, b$ is $(123, 1\text{-})$-avoiding. Using the bijection f described in the previous section, we can thus map each subtree $L(r_i)$ to a Motzkin path $f(L(r_i))$. Therefore, we map T to an $(n-1)$-step Motzkin left factor $g(T)$ by combining the subpaths $f(L(r_i))$, separating them by in total $b-1$ many U-steps, one between every two consecutive subpaths $f(L(r_i))$ and $f(L(r_{i+1}))$. To make the proof work, the subpaths $f(L(r_i))$ can be combined in increasing order from left to right on $g(T)$, i.e., for $i = 1, \ldots, b$, or in decreasing order, i.e., for $i = b, b-1, \ldots, 1$, and for reasons that will become clear in the next section we combine them in decreasing order, i.e.,

$$g(T) := f(L(r_b)), \mathtt{U}, f(L(r_{b-1})), \ldots, \mathtt{U}, f(L(r_1)). \tag{3}$$

The mapping g is clearly a bijection between $\mathcal{T}_n(P,e)$ and $\mathcal{L}_{n-1}$.

6.3 Bijection between $\mathcal{T}_n(21543, \text{-}01\text{-})$ and Motzkin paths with catastrophes $\mathcal{C}_n$

This bijection is illustrated in Figure 13 (c), and it uses as a building block the bijection g defined in the previous section. Instead of $(21543, \text{-}01\text{-})$, we consider the mirrored tree pattern $(P,e) := \mu(21543, \text{-}01\text{-}) = (41235, 01\text{--})$ for convenience. Consider a tree $T \in \mathcal{T}_n(P,e)$ and the rightmost leaf in T, and partition the path from the root of T to that leaf into a sequence of maximal right branches $B_1, \ldots, B_\ell$. For $i = 1, \ldots, \ell$, we let T_i be the subtree of T that consists of B_i plus the left subtrees of all vertices on B_i except the last one. Note that $T_1, \ldots, T_\ell$ form a partition of T. Furthermore, T avoiding (P,e) is equivalent to each of the T_i, $i = 1, \ldots, \ell$, avoiding $(4123, 01\text{-})$. Using the bijection g described in the previous section, we can thus map each subtree T_i to a Motzkin left factor $g(T_i)$, and by appending one additional appropriate step F, D or $\mathtt{D}_h$ for $h \geq 2$ we obtain a Motzkin path $g'(T_i)$. Note that

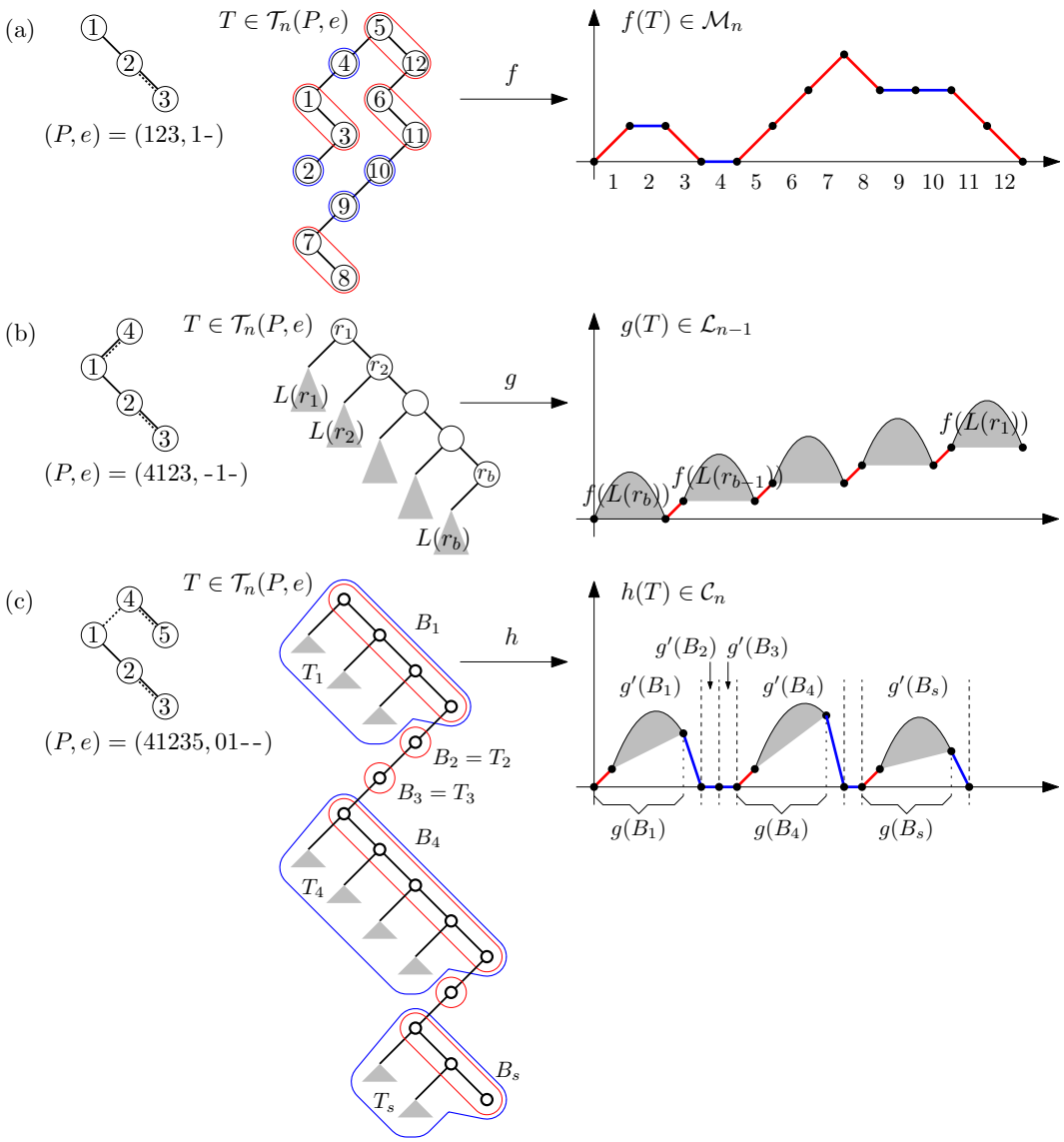


Figure 13 Bijections between pattern-avoiding binary trees and different types of Motzkin paths. Edges $(i, p(i))$ in the tree patterns that can be contiguous or non-contiguous (giving the same pattern-avoiding trees) are drawn as a double line that is half solid and half dotted.

the rightmost leaf of T_i has no left child, and thus the definition (3) yields that $g'(T_i)$ touches the x-axis only at the first point and last point, but at no intermediate (integer) points. Therefore, we map T to an n-step Motzkin path with catastrophes $h(T)$ by concatenating the Motzkin subpaths $g'(T_i)$ for $i = 1, \ldots, \ell$, i.e., $h(T) := g'(T_1), g'(T_2), \ldots, g'(T_\ell)$. It can be readily checked that h is a bijection between $\mathcal{T}_n(P, e)$ and $\mathcal{C}_n$.

7 Open Problems

- Are there elegant bijections between pattern-avoiding binary trees and other interesting combinatorial objects such as Motzkin paths with 2-colored F-steps at odd heights (OEIS A176677), or so-called skew Motzkin paths (OEIS A025242)?

- For purely contiguous or non-contiguous tree patterns (P, e), there are recursions to derive the generating function for $|\mathcal{T}_n(P, e)|$; see [38] and [11]. For our more general mixed tree patterns, these methods seem to fail. Is there is an algorithm to compute those more general generating functions, and what are their properties? Furthermore, can the set of pattern-avoiding trees for such pure (non-friendly) patterns be generated efficiently?
- In addition to contiguous and non-contiguous edges $(i, p(i))$ of a binary tree pattern, which we encode by $e(i) = 1$ and $e(i) = 0$, there is another very natural notion of pattern containment that is intermediate between those two, which we may encode by setting $e(i) := 1/2$. Specifically, for such an edge with $e(i) = 1/2$ in the pattern tree P, we require from the injection f described in Section 2.2 that $f(i)$ is a descendant of $f(p(i))$ along a left or right branch in the host tree T. Specifically, if $i = c_L(p(i))$, then $f(i) = c_L^j(f(p(i)))$ for some $j > 0$, whereas if $i = c_R(p(i))$, then $f(i) = c_R^j(f(p(i)))$ for some $j > 0$. Theorem 2 can be generalized to also capture this new notion, by modifying the definition (2b) in the natural way to

$$C'_i := \begin{cases} \emptyset & \text{if } e(i) = 0, \\ \{(\rho(i) - 1, \min P(i) - 1)\} & \text{if } e(i) = 1/2 \text{ and } i = c_L(p(i)), \\ \{(\rho(i) - 1, \max P(i))\} & \text{if } e(i) = 1/2 \text{ and } i = c_R(p(i)), \\ \{(\rho(i) - 1, \min P(i) - 1), (\rho(i) - 1, \max P(i))\} & \text{if } e(i) = 1. \end{cases}$$

  The notion of friendly tree pattern can be generalized by modifying condition (iii) in Section 4.2 as follows: (iii') If $e(k) \in \{1, 1/2\}$, then we have $e(c_L(k)) \in \{0, 1/2\}$. It is worthwhile to investigate this new notion of pattern containment/avoidance and its interplay with the other two notions. Our computer experiments show that there are patterns with edges $e(i) = 1/2$ that give rise to counting sequences that are distinct from the ones obtained from patterns with edges $e(i) = 1$ (contiguous) and $e(i) = 0$ (non-contiguous). The corresponding functionality has already been built into our generation tool [10].
  This intermediate notion of pattern-avoidance in binary trees has interesting applications in the context of pattern-avoidance in rectangulations, a line of inquiry that was initiated in [34].

References

1. K. Anders and K. Archer. Rooted forests that avoid sets of permutations. *European J. Combin.*, 77:1–16, 2019. doi:10.1016/j.ejc.2018.10.004.
2. A. Asinowski, A. Bacher, C. Banderier, and B. Gittenberger. Analytic combinatorics of lattice paths with forbidden patterns: enumerative aspects. In *Language and automata theory and applications*, volume 10792 of *Lecture Notes in Comput. Sci.*, pages 195–206. Springer, Cham, 2018. doi:10.1007/978-3-319-77313-1_15.
3. E. Babson and E. Steingrímsson. Generalized permutation patterns and a classification of the Mahonian statistics. *Sém. Lothar. Combin.*, 44:Art. B44b, 18 pp., 2000.
4. J.-L. Baril and S. Kirgizov. Bijections from Dyck and Motzkin meanders with catastrophes to pattern avoiding Dyck paths. *Discrete Math. Lett.*, 7:5–10, 2021. doi:10.47443/dml.2021.0032.
5. A. Bernini, L. Ferrari, R. Pinzani, and J. West. Pattern-avoiding Dyck paths. In *25th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2013)*, Discrete Math. Theor. Comput. Sci. Proc., AS, pages 683–694. Assoc. Discrete Math. Theor. Comput. Sci., Nancy, 2013.

6 D. Bevan, D. Levin, P. Nugent, J. Pantone, L. Pudwell, M. Riehl, and M. L. Tlachac. Pattern avoidance in forests of binary shrubs. *Discrete Math. Theor. Comput. Sci.*, 18(2):Paper No. 8, 22 pp., 2016.

7 J. Bloom and S. Elizalde. Pattern avoidance in matchings and partitions. *Electron. J. Combin.*, 20(2):Paper 5, 38, 2013. doi:10.37236/2976.

8 J. Bloom and D. Saracino. Pattern avoidance for set partitions à la Klazar. *Discrete Math. Theor. Comput. Sci.*, 18(2):Paper No. 9, 22 pp., 2016. doi:10.46298/dmtcs.1327.

9 P. Brändén and A. Claesson. Mesh patterns and the expansion of permutation statistics as sums of permutation patterns. *Electron. J. Combin.*, 18(2):Paper 5, 14 pp., 2011.

10 The Combinatorial Object Server: Generate binary trees. http://www.combos.org/btree.

11 M. Dairyko, L. Pudwell, S. Tyner, and C. Wynn. Non-contiguous pattern avoidance in binary trees. *Electron. J. Combin.*, 19(3):Paper 22, 21 pp., 2012. doi:10.37236/2099.

12 F. Disanto. Unbalanced subtrees in binary rooted ordered and un-ordered trees. *Sém. Lothar. Combin.*, 68:Art. B68b, 14 pp., 2012.

13 V. Dotsenko. Pattern avoidance in labelled trees, 2011. arXiv:1110.0844.

14 S. Elizalde and M. Noy. Consecutive patterns in permutations. *Adv. in Appl. Math.*, 30:110–125, 2003. Formal power series and algebraic combinatorics (Scottsdale, AZ, 2001). doi:10.1016/S0196-8858(02)00527-4.

15 N. Gabriel, K. Peske, L. Pudwell, and S. Tay. Pattern avoidance in ternary trees. *J. Integer Seq.*, 15(1):Article 12.1.5, 20 pp., 2012.

16 S. Giraudo. Tree series and pattern avoidance in syntax trees. *J. Combin. Theory Ser. A*, 176:105285, 37, 2020. doi:10.1016/j.jcta.2020.105285.

17 A. Godbole, A. Goyt, J. Herdan, and L. Pudwell. Pattern avoidance in ordered set partitions. *Ann. Comb.*, 18(3):429–445, 2014. doi:10.1007/s00026-014-0232-y.

18 A. M. Goyt. Avoidance of partitions of a three-element set. *Adv. in Appl. Math.*, 41(1):95–114, 2008. doi:10.1016/j.aam.2006.07.006.

19 A. M. Goyt and L. K. Pudwell. Avoiding colored partitions of two elements in the pattern sense. *J. Integer Seq.*, 15(6):Article 12.6.2, 17 pp., 2012.

20 P. Gregor, T. Mütze, and Namrata. Combinatorial generation via permutation languages. VI. Binary trees, 2023. Full preprint version of the present article available at arXiv:2306.08420.

21 E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. I. Fundamentals. *Trans. Amer. Math. Soc.*, 375(4):2255–2291, 2022. doi:10.1090/tran/8199.

22 V. Jelínek and T. Mansour. On pattern-avoiding partitions. *Electron. J. Combin.*, 15(1):Research paper 39, 52 pp., 2008. URL: http://www.combinatorics.org/Volume_15/Abstracts/v15i1r39.html.

23 V. Jelínek, T. Mansour, and M. Shattuck. On multiple pattern avoiding set partitions. *Adv. in Appl. Math.*, 50(2):292–326, 2013. doi:10.1016/j.aam.2012.09.002.

24 M. Klazar. On *abab*-free and *abba*-free set partitions. *European J. Combin.*, 17(1):53–68, 1996. doi:10.1006/eujc.1996.0005.

25 M. Klazar. Counting pattern-free set partitions. I. A generalization of Stirling numbers of the second kind. *European J. Combin.*, 21(3):367–378, 2000. doi:10.1006/eujc.1999.0353.

26 M. Klazar. Counting pattern-free set partitions. II. Noncrossing and other hypergraphs. *Electron. J. Combin.*, 7:Research Paper 34, 25 pp., 2000. URL: http://www.combinatorics.org/Volume_7/Abstracts/v7i1r34.html.

27 G. D. Knott. A numbering system for binary trees. *Commun. ACM*, 20(2):113–115, 1977. doi:10.1145/359423.359434.

28 D. E. Knuth. *The Art of Computer Programming. Vol. 1: Fundamental algorithms*. Addison-Wesley, Reading, MA, 1997. Third edition.

29 G. Kreweras. Sur les partitions non croisées d'un cycle. *Discrete Math.*, 1(4):333–350, 1972. doi:10.1016/0012-365X(72)90041-6.

30 D. Levin, L. K. Pudwell, M. Riehl, and A. Sandberg. Pattern avoidance in k-ary heaps. *Australas. J. Combin.*, 64:120–139, 2016.

31 J. M. Lucas, D. Roelants van Baronaigien, and F. Ruskey. On rotations and the generation of binary trees. *J. Algorithms*, 15(3):343–366, 1993. doi:10.1006/jagm.1993.1045.

32 T. Mansour and M. Shattuck. Pattern avoiding partitions and Motzkin left factors. *Cent. Eur. J. Math.*, 9(5):1121–1134, 2011. doi:10.2478/s11533-011-0057-4.

33 T. Mansour and M. Shattuck. Pattern avoiding partitions, sequence A054391 and the kernel method. *Appl. Appl. Math.*, 6(12):397–411, 2011.

34 A. Merino and T. Mütze. Combinatorial generation via permutation languages. III. Rectangulations. *Discrete Comput. Geom.*, 70:51–122, 2023. doi:10.1007/s00454-022-00393-w.

35 T. Mütze. Combinatorial Gray codes – an updated survey. *Electron. J. Combin.*, DS26:93, 2023. doi:10.37236/11023.

36 OEIS Foundation Inc. The on-line encyclopedia of integer sequences, 2023. URL: http://oeis.org.

37 L. Pudwell, C. Scholten, T. Schrock, and A. Serrato. Noncontiguous pattern containment in binary trees. *International Scholarly Research Notices*, 2014, 2014. doi:10.1155/2014/316535.

38 E. S. Rowland. Pattern avoidance in binary trees. *J. Combin. Theory Ser. A*, 117(6):741–758, 2010. doi:10.1016/j.jcta.2010.03.004.

39 B. E. Sagan. Pattern avoidance in set partitions. *Ars Combin.*, 94:79–96, 2010.

40 A. Sapounakis, I. Tasoulas, and P. Tsikouras. Counting strings in Dyck paths. *Discrete Math.*, 307(23):2909–2924, 2007. doi:10.1016/j.disc.2007.03.005.

41 C. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, 1997. doi:10.1137/S0036144595295272.

42 A. Williams. The greedy Gray code algorithm. In *Algorithms and data structures*, volume 8037 of *Lecture Notes in Comput. Sci.*, pages 525–536. Springer, Heidelberg, 2013. doi:10.1007/978-3-642-40104-6_46.

Computing a Subtrajectory Cluster from c-Packed Trajectories

Joachim Gudmundsson
The University of Sydney, Australia

Zijin Huang
The University of Sydney, Australia

André van Renssen
The University of Sydney, Australia

Sampson Wong
BARC, University of Copenhagen, Denmark

Abstract

We present a near-linear time approximation algorithm for the subtrajectory cluster problem of c-packed trajectories. Given a trajectory T of complexity n, an approximation factor ε, and a desired distance d, the problem involves finding m subtrajectories of T such that their pair-wise Fréchet distance is at most $(1+\varepsilon)d$. At least one subtrajectory must be of length l or longer. A trajectory T is c-packed if the intersection of T and any ball B with radius r is at most $c \cdot r$ in length.

Previous results by Gudmundsson and Wong [24] established an $\Omega(n^3)$ lower bound unless the Strong Exponential Time Hypothesis fails, and they presented an $O(n^3 \log^2 n)$ time algorithm. We circumvent this conditional lower bound by studying subtrajectory cluster on c-packed trajectories, resulting in an algorithm with an $O((c^2 n/\varepsilon^2)\log(c/\varepsilon)\log(n/\varepsilon))$ time complexity.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Subtrajectory cluster, c-packed trajectories, Computational geometry

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.34

Related Version *Full Version*: `https://arxiv.org/abs/2307.10610`

Funding *Joachim Gudmundsson*: Funded by the Australian Government through the Australian Research Council DP180102870.

Acknowledgements The authors thank Kevin Buchin for the insightful discussion on the important Lemma 13. The authors thank the anonymous reviewers for their helpful feedback.

1 Introduction

With the proliferation of location-aware devices comes an abundance of trajectory data. One way to process and make sense of many trajectories is to group long and similar subtrajectories. The analysis of long and similar parts of trajectories can provide insights into behavior and mobility patterns, such as common routes taken and places visited frequently.

Buchin et al. [8] initialised the study of subtrajectory cluster problems to detect and extract common movement patterns. The Subtrajectory Cluster (SC) decision problem is defined as follows. Given one or more trajectories, determine if there exists a cluster of $m-1$ non-overlapping subtrajectories and one reference trajectory. The reference trajectory T_r must be at least of length l, and the Fréchet distances between T_r and the other $m-1$ subtrajectories must be at most d. In the case of animals, long and common movement patterns can indicate movement between grazing spots of sheep or the migration flyway of seabirds. In the case of humans, common movement on a Monday morning can show commuting patterns to find the most heavily congested areas.

Subtrajectory clustering has attracted research from multiple communities. Gudmundsson and Wolle [23] used subtrajectory cluster to analyse the common movement patterns of football players. Buchin et al. [7] applied subtrajectory cluster to map reconstruction by clustering common movement patterns of vehicles into road segments. Researchers in the Geographical Information and Data Mining communities also considered the variants and practical performance of subtrajectory cluster algorithms [11, 29, 21, 22, 26, 1, 17]. In addition, the potential of SC is examined in a wide range of applications, including sports player analysis [30] and human movement analysis [10, 25].

Several theoretical studies of the subtrajectory clustering problem focus on improving the quality of clustering. Agarwal et al. [1] defined a single objective function, the weighted sum of three quality measures of a clustering. These quality measures include the number of clusters chosen, the quality of the cluster, and the size of the trajectories excluded from the clustering. Brüning et al. [6] studied so-called $\triangle$-coverage, aiming to find a set C of curves to cover a polygonal curve such that a curve in C is fixed in size, and $|C|$ is minimised.

However, despite considerable attention from multiple communities, there is no subcubic time algorithm that solves the subtrajectory cluster problem, limiting its usefulness on large data sets. Buchin et al. [8] solved the subtrajectory cluster problem in $O(n^5)$ time when the similarity measurement of two trajectories is the Fréchet distance, and Gudmundsson and Wong [24] further improved the runtime with an $O(n^3 \log^2 n)$ time algorithm. In addition, Gudmundsson and Wong [24] showed that there is no $O(n^{3-\delta})$ algorithm for subtrajectory cluster for any $\delta > 0$ unless the Strong Exponential Time Hypothesis (SETH) fails.

SC is unlikely to have a strongly subquadratic algorithm even if we allow a small approximation factor on the Fréchet distances between subtrajectories. Because given two trajectories T_1 and T_2, we can structure the SC problem to find two subtrajectories such that their Fréchet distance is at most $(1+\varepsilon)d$, and the reference trajectory must be as long as the maximum of T_1 and T_2. Solving this instance of SC is equivalent to approximating the Fréchet distance of T_1 and T_2, and Bringmann [4] showed that there is no 1.001-approximation with runtime $O(n^{2-\delta})$ for the continuous Fréchet distance for any $\delta > 0$, unless SETH fails.

Since an exact subcubic and an approximate subquadratic algorithm are unlikely to exist, we study subtrajectory cluster on a realistic family of trajectories, called c-packed trajectories. A trajectory T is c-packed if, for any ball B of radius r, the length of T lying inside B is at most c times r. The packedness value of a trajectory T is the maximum c for which T is c-packed. Bringmann [4] proved that computing the Fréchet distance has no strong subquadratic algorithm unless SETH fails, and the notion of c-packedness was introduced by Driemel et al. [15] to circumvent such conditional lowerbound. Since then, the notion of c-packedness has gained considerable attention from the theory community [20, 4, 2, 12, 5], and several real-world data sets have been shown to have low packedness values [19, 15]. In one particular instance, Gudmundsson et al. [19] approximated the packedness values of several real-world trajectory data sets. In their experiments, several trajectory data sets have low packedness values, such as the movement patterns of people in Beijing, school buses, European football players, and trawling bats.

In this paper, given a c-packed trajectory T of complexity n and a desired multiplicative approximation error ε on the Fréchet distance between subtrajectories, we present an $O((c^2 n/\varepsilon^2)\log(c/\varepsilon)\log(n/\varepsilon))$ time algorithm that solves the SC problem. It is worth noting that previous papers considering c-packed curves typically replace a factor n with a polynomial of constant degree in c [13, 18]. We are able to replace a factor of n^2 with c^2/ε^2, bringing the algorithm's running time from cubic to near-linear, assuming $c \in O(1)$.

Along the way, we develop a tool for simplifying the free space diagram that may be of independent interest. To efficiently approximate the Fréchet distance, Driemel et al. [15] showed that the free space complexity, i.e., the number of non-empty cells, is $O(cn/\varepsilon)$ for two simplified c-packed trajectories (see Section 2 for an overview of the free space, or [3] for a formal definition). However, simplifying a trajectory by taking shortcuts between vertices can yield a much shorter trajectory, and the SC problem is sensitive to the length of the trajectories since the reference trajectory has to have a length at least l. To tackle this problem, we developed a tool to construct the free space diagram in $O((cn/\varepsilon)\log(cn/\varepsilon))$ time, preserving the length of two trajectories while benefiting from the $O(cn/\varepsilon)$ free space complexity. Our tool can be of value for problems in which the length of a trajectory is important, such as subtrajectory cluster [8], partial curve matching [9], and Fréchet distance with speed limit [27].

In Section 2, we will formally define the subtrajectory cluster problem and outline the greedy plane sweep algorithms by Buchin et al. [8] and Gudmundsson and Wong [24], which our approach builds on. In Section 3, we provide a technical overview of our main results. In Section 4, we will discuss how to simplify the free space diagram to achieve a lower complexity while preserving trajectory lengths.

In Section 5, we will consider the restricted case when the reference trajectory must be vertex-to-vertex. In Section 6, we will remove this restriction by considering arbitrary reference trajectory.

2 Preliminaries

In this section, we will outline the previous algorithms for the subtrajectory cluster problem. The subtrajectory cluster problem was first introduced by Buchin et al. [8], and later improved by Gudmundsson and Wong [24]. But instead of looking for a subtrajectory where the Fréchet distances between the reference trajectory and the subtrajectories are exact, we aim to find a solution that approximates the Fréchet distance between subtrajectories in the cluster.

▶ **Problem 1** ([24]). *Given trajectory T of complexity n, a positive integer m, positive real numbers d, l and ε, decide if there exists a subtrajectory cluster of T such that:*

- *the cluster consists of one reference subtrajectory and $m-1$ other subtrajectories of T,*
- *the reference subtrajectory has Euclidean length at least l,*
- *the Fréchet distance between the reference subtrajectory and any other subtrajectory is at most $(1+\varepsilon)d$,*
- *any pair of subtrajectories in the cluster overlap in at most one point.*

Buchin et al. [8] solved the exact SC problem by using a plane sweep algorithm on the free space diagram $\mathcal{F}_d(T,T)$. Let s and t be two points on T, and we denote T_{st} as the subtrajectory of T starting from s and ending on t. Let l_s and l_t be the vertical sweep lines $x=s$ and $x=t$ on $\mathcal{F}_d(T,T)$, respectively (see Figure 1). An xy-monotone path in $\mathcal{F}_d(T,T)$, or monotone path for short, is a continuous path that is non-decreasing in both x- and y-coordinates. To solve `SC(m, d, l)`, the lines l_s and l_t sweep from left to right while making sure that l_s is to the left of l_t, and the reference trajectory T_{st} is at least l long, i.e., $t-s \geq l$. In each interval $[l_s, l_t]$, they compute the maximum number of monotone paths in $\mathcal{F}_d(T,T)$ starting at l_s and ending at l_t.

Let p and q be two points on T, and let (s,p) and (t,q) be two coordinates on $\mathcal{F}_d(T,T)$. As we only consider monotone paths starting from l_s and ending on l_t, we call the monotone path from (s,p) to (t,q) the pq monotone path. First, a monotone path pq must traverse only

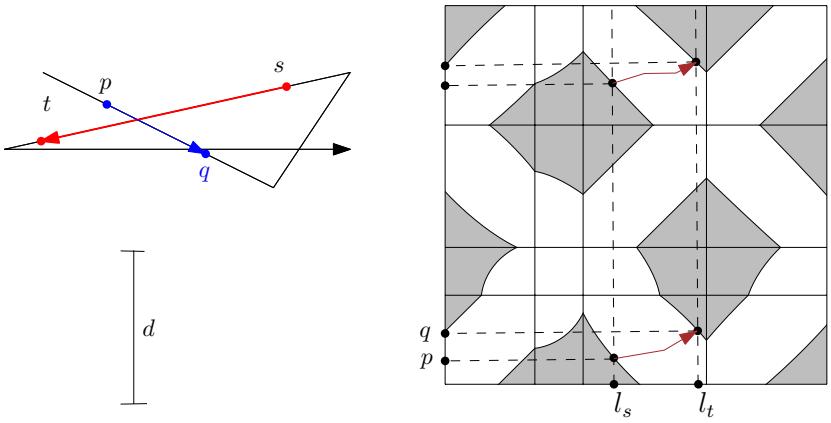


Figure 1 The free space diagram $\mathcal{F}_d(T,T)$, and interval $[l_s, l_t]$ defined by two points s and t on T. If there exists a monotone path (marked in brown) from (s,p) to (t,q) through the free space, then the Fréchet distance between subtrajectories T_{st} and T_{pq} is at most d.

the free space. Second, two monotone paths pq and ab must not overlap along the y-interval in more than a single point. Third, the y-coordinates of any pq monotone path cannot overlap the $[s,t]$ interval in more than a single point. We obtain the following subproblem.

▶ **Subproblem 2** ([24]). *Given a trajectory T of complexity n, a positive integer m, a positive real value d, and a reference subtrajectory of T starting at s and ending at t, let l_s and l_t be two vertical lines in $\mathcal{F}_d(T,T)$ representing the points s and t. Decide if there exist:*

- *$m-1$ distinct paths starting at l_s and ending at l_t, such that*
- *the y-coordinate of any two monotone paths overlap in at most one point, and*
- *the y-coordinate of any monotone path overlaps the y-interval from s to t in at most one point.*

To look for a set $\{p_1q_1, p_2q_2, ..., p_{m-1}q_{m-1}\}$ of monotone paths, both algorithms use a greedy approach. First, set p_1 to be the lowest feasible point on l_s, and compute p_1q_1 by searching for a lowest monotone path through the free space. Inductively, with $p_{i-1}q_{i-1}$ computed, set p_i to the lowest feasible point on l_s that is on or above q_{i-1}, and do the same. If a search from p_i leads to a dead end, we simply set p_i to the next lowest feasible point on l_s, and search again.

The sweeplines stop at all $O(n^3)$ critical points, and for each critical point there is a $[l_s, l_t]$ interval to consider. Buchin et al. [8] solved each instance in $O(nm) \subseteq O(n^2)$ time. Gudmundsson and Wong [24] improved the efficiency by connecting the critical points efficiently in a tree-like data structure which allows them to reuse computed monotone paths from previous interval instances. They showed that, in their construction, there are at most $O(n^3 \log n)$ edges, and each edge takes at most $O(\log n)$ time to add, remove, or access. This brings down the complexity of the algorithm from $O(n^5)$ to $O(n^3 \log^2 n)$ time.

3 Technical Overview

Our technical overview is divided into three parts. In Sections 3.1, 3.2, and 3.3, we summarise the main result of Sections 4, 5, and 6 respectively.

3.1 Computing the Free Space Diagram

Our algorithm constructs a simplified free space diagram that preserves trajectory lengths. The size (in terms of Euclidean length) of the simplified free space diagram is the same as the size of the unsimplified free space diagram. The only difference between the two diagrams is that approximate distances are used in the simplified diagram. In particular, we define a function that uniformly maps a trajectory to its simplification, and we calculate the distance between the mapped simplification points instead of points on the original trajectory. We prove that the complexity of the simplified free space diagram will be at most $O(cn/\varepsilon)$, and that the trajectory lengths in the diagram are preserved. Next, we build the simplified free space diagram. We use an algorithm by Conradi and Driemel [13] to query pairs of nearby segments. Finally, we construct a data structure on the free space diagram so that we can access the closest non-empty cells below, above, to the left, and to the right in constant time. Putting this all together, we obtain Theorem 1. For a full proof, see Section 4.

▶ **Theorem 1.** *Given a pair of trajectories, one can construct a simplified free space diagram in $O((cn/\varepsilon)\log(cn/\varepsilon))$ time, so that the simplified free space has complexity $O(cn/\varepsilon)$, it approximates the Fréchet distance to within a factor of $(1+\varepsilon)$, and it preserves the trajectory lengths of the original trajectory.*

3.2 Reference trajectory is vertex-to-vertex

Next, we focus on the special case where the reference trajectory is vertex-to-vertex. Three data structures are used in the vertex-to-vertex subtrajectory cluster algorithm of Gudmundsson and Wong [24] – a directed graph, a range tree, and a link-cut tree. For an overview of these data structures, see the full version. Originally, the number of leaves per range tree is $O(n)$, and the directed graph has complexity $O(n^2)$. We use the c-packedness property to prove that, in our simplified free space diagram, the number of leaves per range tree is $O(c/\varepsilon)$, and the directed graph has complexity $O((cn/\varepsilon)\log(c/\varepsilon))$. The link-cut tree data structure can be used without modification. Putting this all together, we obtain Theorem 2. Recall that m is the desired number of subtrajectories in the cluster. For a full proof, see Section 5.

▶ **Theorem 2.** *There is an $O(nm\log(c/\varepsilon)\log(n/\varepsilon))$ time algorithm that solves $\mathtt{SC}(\mathtt{T},\mathtt{m},\mathtt{l},(\mathtt{1}+\varepsilon)\mathtt{d})$ in the case that the reference trajectory is vertex-to-vertex.*

3.3 Reference trajectory is arbitrary

Finally, we tackle the general case where the reference trajectory is arbitrary. The main obstacle in the general case is that there are $\Theta(n^3)$ internal critical points that correspond to potential starting and ending positions of the reference trajectory. In fact, Gudmundsson and Wong [24] show that, for general (not c-packed) curves, these internal critical points are essentially unavoidable. They use the $\Theta(n^3)$ internal critical points to show that under the Strong Exponential Time Hypothesis (SETH), there is no $O(n^{3-\delta})$ time algorithm for subtrajectory cluster for any $\delta > 0$.

Our main lemma in this section is to bound the number of internal critical points for subtrajectory cluster on c-packed trajectories. The lemma uses the c-packedness property in two different ways. First, the c-packedness property bounds the complexity of the simplified free space diagram to linear. This replaces one of the factors of n with c/ε. Second, the c-packedness property is used to prove that in any horizontal strip, only a constant number of cells have free space. This replaces another factor of n with c/ε, resulting in $O(c^2n/\varepsilon^2)$

internal critical points. Finally, we prove that the interval management data structure can be used in the same way as in Gudmundsson and Wong's algorithm [24]. Putting this all together, we obtain Theorem 3. For a full proof, see Section 6.

▶ **Theorem 3.** *There is an* $O((c^2n/\varepsilon^2)\log(c/\varepsilon)\log(n/\varepsilon))$ *time algorithm that solves* $\mathtt{SC}(\mathtt{T}, \mathtt{m}, \mathtt{l}, (\mathtt{1}+\varepsilon)\mathtt{d})$ *in the case that the reference trajectory is arbitrary.*

4 Computing the Free Space Diagram

In this section, we will explain the process of constructing a simplified free space diagram for two c-packed polygonal curves P and Q. The free space $\mathcal{D}_d(P,Q)$ describes all pairs of points, one on P, one on Q, whose distance is at most d [3]. With slight abuse of notation, we parameterise the polygonal curve P such that $P(x)$ is a point on P, where $x \in [0, \|P\|]$. Formally,

$$\mathcal{D}_d(P,Q) = \{(x,y) \in [0,\|P\|] \times [0,\|Q\|] \mid dist(P(x),Q(y)) \leq d\}.$$

To circumvent the quadratic free space complexity, Driemel et al. [15] showed that the free space complexity of two simplified c-packed curves is $O(cn/\varepsilon)$. Given a c-packed curve $P = p_1p_2...p_n$, we simplify P into its εd-simplification $P' = q_1q_2...q_k$ as follows. Let $B(a,r)$ be the ball centered at a with radius r. First, set $q_1 = p_1$. With q_i defined, traverse P from q_i until a vertex v is outside $B(q_i, \varepsilon d)$, or v is the last vertex of P, and set $q_{i+1} = v$. Continue until all vertices of P are exhausted. Driemel et al. [15] showed that the εd-simplification of a c-packed curve is at most $6c$-packed [15, Lemma 4.3], and the Fréchet distance between P and P' is at most $(1+\varepsilon)d$. A simplified curve has the useful property that every segment but the last is at least εd long. We assume for simplicity that the last one is at least εd long as well, since otherwise one can backtrack and modify the simplified curve such that each segment is at least $\varepsilon d/2$ long, and our arguments can be extended to such case.

▶ **Observation 4.** *One can simplify a polygonal curve* P *into its* εd*-simplification* P' *such that the Fréchet distance between* P *and* P' *is at most* $(1+\varepsilon)d$*, and each segment in* P' *is at least* εd *long.*

Simplifying two c-packed curves can reduce the free space complexity, but using the plane-sweep algorithm to solve the SC problem on the resulting free space diagram is unfortunately infeasible. This is because the total length of the simplified trajectories can be much shorter, making it impossible to slide a window of width l on the free space diagram $\mathcal{F}_{(1+\varepsilon)d}(P',Q')$. To address this issue, we developed a tool that enables the construction of a free space diagram that maintains the original curve length, while also benefiting from the reduced free space complexity.

4.1 Simplifying the Free Space

In this section, we introduce a method that simplifies the free space. We show that we can construct the *simplified free space* $\mathcal{D}'_{(1+\hat{\varepsilon})d}(P,Q)$, where $\hat{\varepsilon}$ is at most 8ε, such that the complexity of the simplified free space is at most $O(cn/\hat{\varepsilon})$. In addition, $\mathcal{D}'_{(1+\hat{\varepsilon})d}(P,Q)$ contains $\mathcal{D}_d(P,Q)$ as a subset, but it is not bigger than the free space of P and Q if we approximate their Fréchet distance, i.e., $\mathcal{D}'_{(1+\hat{\varepsilon})d}(P,Q) \subseteq \mathcal{D}_{(1+\hat{\varepsilon})d}(P,Q)$.

We will first define a function that uniformly maps parts of the polygonal curve P to segments of P' in Definition 5, using which we will formally define the simplified free space in Definition 6. We will then formally prove the set inclusions mentioned above in Lemma 7.

▶ **Definition 5.** *Let $simpl(P, \varepsilon d)$ be the εd-simplification of a polygonal curve P. Let P_{uv} be the subcurve of P from point u to v that are simplified into the segment $(u, v) \in simpl(P, \varepsilon d)$. Let w be the first intersection point of P_{uv} and the boundary of the ball $B(u, \varepsilon d)$ along P_{uv}, and let u' be the intersection of (u, v) and the boundary of the ball $B(u, \varepsilon d)$. Define the mapping $f_{P,\varepsilon d} : P \to simpl(P, \varepsilon d)$ such that $f_{P,\varepsilon d}$ maps $[u, w)$ to $[u, u')$ uniformly, and $[w, v]$ to $[u', v]$ uniformly (see Figure 2).*

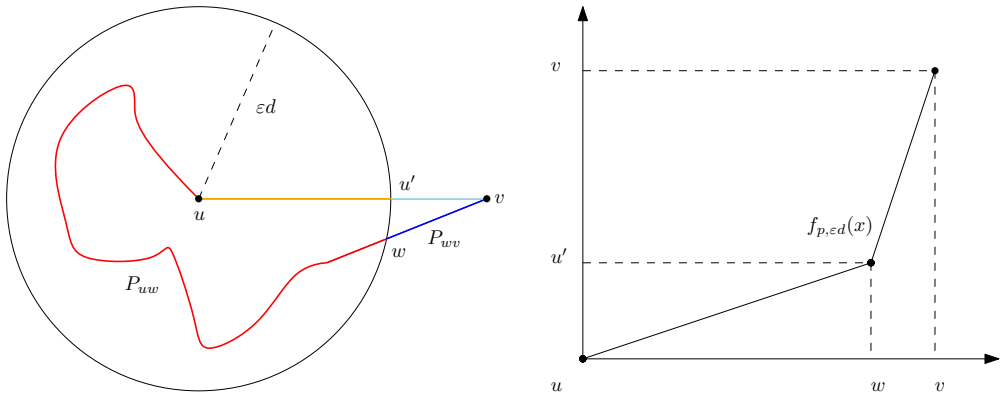


Figure 2 A figure showcasing the function in Definition 5. The point u' is the intersection of the segment (u, v) and the ball $B(u, \varepsilon d)$, and the point w is the intersection of subtrajectory P_{uv} and $B(u, \varepsilon d)$. The function $f_{P,\varepsilon d}$ uniformly maps P_{uw} (red) to (u, u') (orange), not including u' and w. The function $f_{P,\varepsilon d}$ uniformly maps P_{wv} (blue) to (u', v) (light blue).

▶ **Definition 6.** *Define the simplified free space of P and Q with respect to the Fréchet distance $d > 0$, and a parameter $\varepsilon > 0$ as*

$$\mathcal{D}'_{(1+\varepsilon)d}(P, Q) = \{(x, y) \in [0, \|P\|] \times [0, \|Q\|] \mid dist(f_{P,\varepsilon d}(P(x)), f_{Q,\varepsilon d}(Q(y))) \leq (1 + \varepsilon)d\}.$$

Similarly, let $\mathcal{F}'_{(1+\varepsilon)d}(P, Q)$ be the simplified free space diagram.

▶ **Lemma 7.** *Let $\mathcal{D}_d(P, Q)$ be the free space of curves P and Q with respect to the Fréchet distance d, and let $\mathcal{D}'_{(1+\varepsilon)d}(P, Q)$ be their simplified free space with an approximation error $\varepsilon > 0$. Then $\mathcal{D}_d(P, Q) \subseteq \mathcal{D}'_{(1+4\varepsilon)d}(P, Q) \subseteq \mathcal{D}_{(1+8\varepsilon)d}(P, Q)$.*

Proof. With slight abuse of notation, let $x = P(x)$, and $y = Q(y)$, for $x \in [0, \|P\|]$, and $y \in [0, \|Q\|]$. Let $x' = f_{P,\varepsilon d}(x)$, and let $y' = f_{Q,\varepsilon d}(y)$. Observe that $dist(x, x') \leq 2\varepsilon d$ for all $x \in P$ because if x is within the ball $B(u, \varepsilon d)$, then x is at most $2\varepsilon d$ apart from x'. If x is outside $B(u, \varepsilon d)$, it is at most εd apart from x', due to the simplification.

- $\mathcal{D}_d(P, Q) \subseteq \mathcal{D}'_{(1+4\varepsilon)d}(P, Q)$. If a point $(x, y) \in \mathcal{D}_d(P, Q)$ is white, then $dist(x, y) \leq d$. By the triangle inequality, $dist(x', y') \leq dist(x', x) + dist(y', y) + dist(x, y) \leq 2\varepsilon d + 2\varepsilon d + d = (1 + 4\varepsilon)d$, hence (x', y') must also be white.
- $\mathcal{D}'_{(1+4\varepsilon)d}(P, Q) \subseteq \mathcal{D}_{(1+8\varepsilon)d}(P, Q)$. Similarly, if a point $(x', y') \in \mathcal{D}'_{(1+4\varepsilon)d}(P, Q)$ is white, then $(x, y) \in \mathcal{D}_{(1+8\varepsilon)d}(P, Q)$ must also be white, because $dist(x, y) \leq dist(x', x) + dist(y', y) + dist(x', y') \leq 2\varepsilon d + 2\varepsilon d + (1 + 4\varepsilon)d = (1 + 8\varepsilon)d$. ◀

Similar to how we defined the (u, v) cell, let the (P_{uv}, Q_{ab}) cells be the cells in the free space diagram defined by the subcurves P_{uv} and Q_{ab}. We show that we can compute the intersection of the simplified free space with (P_{uv}, Q_{ab}) cells in constant time. Please see the full version for a proof of below lemma.

▶ **Lemma 8.** *Given vertices u, v on P and a, b on Q, one can construct the cells in $\mathcal{F}'_{(1+\varepsilon)d}(P, Q)$ defined by P_{uv} and Q_{ab} in constant time.*

The complexity of the simplified free space $\mathcal{D}'_{(1+\hat{\varepsilon})d}(P, Q)$ is $O(cn/\hat{\varepsilon})$ if P and Q are c-packed. Assuming that P_{uv} and Q_{ab} are simplified into segments $(u, v) \in P'$ and $(a, b) \in Q'$, respectively, the simplified free space intersects (P_{uv}, Q_{ab}) cells if and only if the distance between (u, v) and (a, b) is at most $(1 + \hat{\varepsilon})d$. The rest follows by modifying the proof of [15, Lemma 4.4].

▶ **Corollary 9.** *Let P and Q be two c-packed curves with complexity n, and let $\hat{\varepsilon}$ be a constant times a parameter $\varepsilon > 0$. The complexity of the simplified free space $\mathcal{D}'_{(1+\hat{\varepsilon})d}(P, Q)$ is $O(cn/\varepsilon)$.*

4.2 Compute the Non-empty Cells

To take advantage of the near-linear complexity of the simplified free space, we use an algorithm by Conradi and Driemel [13] to efficiently compute the non-empty cells without inspecting all pairs of segments.

▶ **Fact 10** ([13, Lemma 59]). *Given two c-packed curves P and Q in $\mathbb{R}^2$, a parameter $d \geq 0$, and let P' and Q' be their εd-simplifications. In $O((cn/\varepsilon)\log(cn/\varepsilon))$ time, one can find all pairs of segments $(u, v) \in P'$ and $(a, b) \in Q'$ such that the distance between (u, v) and (a, b) is at most d.*

To construct the simplified free space diagram efficiently, we first observe the following.

▶ **Observation 11.** *If segments $(u, v) \in P'$ and $(a, b) \in Q'$ are more than $(1 + 2\varepsilon)d$ apart, then P_{uv} and Q_{ab} are more than d apart.*

The above observation enables us to determine if (P_{uv}, Q_{ab}) cells are empty by determining if (u, v) and (a, b) are near.

4.3 Constructing the Simplified Free Space Diagram

Given two c-packed polygonal curves P and Q, we will use the results from previous subsections to construct the simplified free space diagram using the below steps. In Lemma 8, we showed that if P_{uv} and Q_{ab} are simplified into segments $(u, v) \in P'$ and $(a, b) \in Q'$, respectively, we can compute (P_{uv}, Q_{ab}) cells in constant time. Such aggregation of (P_{uv}, Q_{ab}) cells is an *aggregated non-empty cell*, and we will treat them as one cell for simplicity.

1. Simplify P and Q into their εd-simplifications P' and Q'.
2. Find all pairs of nearby segments from P' and Q' that are at most $(1 + \hat{\varepsilon})d$ apart using Fact 10.
3. For each pair of nearby segments $(u, v) \in P'$ and $(a, b) \in Q'$, compute the (P_{uv}, Q_{ab}) cells using Lemma 8.
4. Sort all non-empty cells horizontally and vertically.
5. Connect non-empty cells in a graph fashion such that a non-empty cell is connected to the first non-empty cells to its top, bottom, left, and right.

Given two polygonal curves P and Q of complexity n, simplifying them (step 1) takes $O(n)$ time. By Fact 10, step 2 takes $O((cn/\varepsilon)\log(cn/\varepsilon))$ time. Computing a cell in $\mathcal{F}'_{(1+\hat{\varepsilon})d}(P, Q)$ takes $O(1)$ time by Lemma 8. $\mathcal{F}'_{(1+\hat{\varepsilon})d}(P, Q)$ has at most $O(cn/\varepsilon)$ non-empty cells, which takes $O(cn/\varepsilon)$ time to compute in step 3; sorting them in step 4 takes $O((cn/\varepsilon)\log(cn/\varepsilon))$ time. Connecting each cell to at most four other cells takes $O(cn/\varepsilon)$ time in step 5. Putting this together, we obtain Lemma 12, and we summarise our result in Theorem 1.

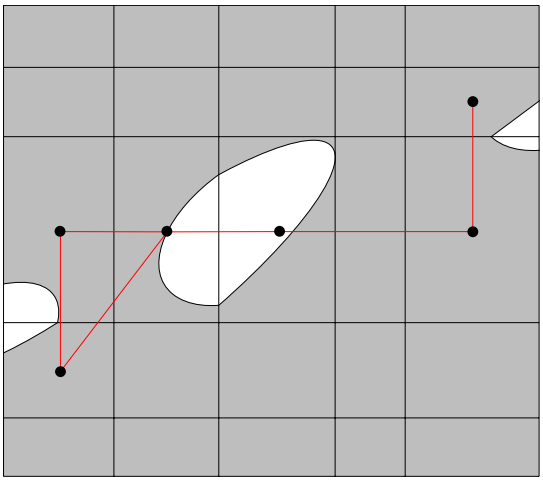


Figure 3 The non-empty cells are connected horizontally and vertically to skip empty cells.

▶ **Lemma 12.** *Let P and Q be two c-packed curves of complexities n. Let $\varepsilon > 0$ and $d > 0$ be two parameters, and let $\hat{\varepsilon} \leq 8 \cdot \varepsilon$. One can construct and connect $O(cn/\varepsilon)$ aggregated non-empty cells of the simplified free space diagram $\mathcal{F}'_{(1+\hat{\varepsilon})d}(P,Q)$ in $O((cn/\varepsilon)\log(cn/\varepsilon))$ time such that $\mathcal{D}_d(P,Q) \subseteq \mathcal{D}'_{(1+\hat{\varepsilon})d}(P,Q) \subseteq \mathcal{D}_{(1+\hat{\varepsilon})d}(P,Q)$. Given an aggregated non-empty cell C, one can access the first aggregated non-empty cells below, above, to the left, and to the right of C in $O(1)$ time.*

▶ **Theorem 1.** *Given a pair of trajectories, one can construct a simplified free space diagram in $O((cn/\varepsilon)\log(cn/\varepsilon))$ time, so that the simplified free space has complexity $O(cn/\varepsilon)$, it approximates the Fréchet distance to within a factor of $(1+\varepsilon)$, and it preserves the trajectory lengths of the original trajectory.*

5 Reference trajectory is vertex-to-vertex

Throughout the rest of the paper we assume that the free space diagram is the simplified free space diagram $\mathcal{F}'_{(1+\varepsilon)d}$ in Lemma 12. Next, we will use the algorithm by Gudmundsson and Wong [24] to determine whether there is a solution to $\mathtt{SC}(\mathtt{T}, \mathtt{m}, \mathtt{1}, (\mathtt{1}+\varepsilon)\mathtt{d})$ where T is a c-packed trajectory, and the reference subtrajectory T_{st} is vertex-to-vertex, i.e., both s and t must be an endpoint of some segment of T.

Three data structures are used in the vertex-to-vertex subtrajectory cluster algorithm of Gudmundsson and Wong [24] – a directed graph, a range tree, and a link-cut tree. For an overview of these data structures, see the full version. In Section 5.1, we show that the number of leaves per range tree is $O(c/\varepsilon)$, and the directed graph has complexity $O((cn/\varepsilon)\log(c/\varepsilon))$. In Section 5.2, we show that the link-cut tree data structure can be used without modification.

5.1 Using a Directed Graph to Store Candidate Monotone Paths

To show that the range tree has at most $O(c/\varepsilon)$ leaves, it suffices to show that there exist at most $O(c/\varepsilon)$ critical points on each horizontal or vertical boundary of the simplified free space diagram.

▶ **Lemma 13.** *In the simplified free space diagram $\mathcal{F}'_{(1+\hat{\varepsilon})d}(T,T)$, let H be a horizontal (resp. vertical) strip that is at least εd wide on its y-span (resp. x-span). The intersection of H and the simplified free space $\mathcal{D}'_{(1+\hat{\varepsilon})d}(T,T)$ exists in at most $O(c/\varepsilon)$ aggregated cells.*

Proof. Let T' be the εd-simplification of T, and let T_{uv} simplifies into segment $(u, v) \in T'$. Let $u' \subseteq (u, v)$ be a small part that is at least εd long. Let $S_{u'} = u' \oplus B(0, (1 + \hat{\varepsilon})d)$.

Using similar construction, and arguments of [15, Lemma 4.4], one can prove that at most $O(c/\varepsilon)$ segments in T' intersects $S_{u'}$. Based on the construction of the simplified free space $\mathcal{D}'_{(1+\hat{\varepsilon})d}(T, T)$, a point $(x, y) \in \mathcal{D}'_{(1+\hat{\varepsilon})d}(T, T)$ is white if and only if $dist(f_{T,\hat{\varepsilon}d}(T(x)), f_{T,\hat{\varepsilon}d}(T(y))) \leq (1 + \hat{\varepsilon})d$. As such, at most $O(c/\varepsilon)$ aggregated cells have simplified free space intersecting H. ◀

Next, bound the construction time and space complexity of the directed graph in [24].

▶ **Lemma 14.** *Given a c-packed trajectory T of complexity n, constructing G for the simplified free space diagram $\mathcal{F}'_{(1+\hat{\varepsilon})d}(T, T)$ takes $O((cn/\varepsilon)\log(n/\varepsilon))$ time. G has $O(cn/\varepsilon)$ nodes and $O((cn/\varepsilon)\log(c/\varepsilon))$ edges.*

Proof. Let n_k be the number of non-empty aggregated cells in the jth row in $\mathcal{F}'_{(1+\hat{\varepsilon})d}(T, T)$. Construction of the range tree for the top (resp. right) boundary of a row (resp. column) takes $O(n_k \log n_k)$ time [14]. For all p_i, finding q_i takes $O(n_k \log n_k)$ time

and recall that there are $O(cn/\varepsilon)$ critical points in $\mathcal{F}'_{(1+\hat{\varepsilon})d}(T, T)$. The total construction time is as follows.

$$\sum_{j=0}^{n+1} n_k \log n_k \leq \log\left(\frac{cn}{\varepsilon}\right) \sum_{j=0}^{n+1} n_k = \log\left(\frac{cn}{\varepsilon}\right) O\left(\frac{cn}{\varepsilon}\right) \in O\left(\left(\frac{cn}{\varepsilon}\right) \log\left(\frac{n}{\varepsilon}\right)\right)$$

By Corollary 9, the simplified free space diagram has $O(cn/\varepsilon)$ non-empty aggregated cells, therefore G has $O(cn/\varepsilon)$ nodes. In a range tree, given a continuous interval $[q_k, q_i]$, one can find $O(\log n)$ nodes such that these nodes include $[q_k, q_i]$ in their canonical subset, where n is the total number of items in the leaves [14]. There are at most $O(c/\varepsilon)$ nodes on a horizontal or vertical boundary by Lemma 13, and each critical point p_i on a vertical (resp. horizontal) cell boundary connects to $O(\log(c/\varepsilon))$ nodes, therefore the total number of edges is $O((cn/\varepsilon)\log(c/\varepsilon))$. ◀

5.2 Storing and Reusing Pre-computed Paths

A link-cut tree [28] maintains a forest that allows the link/cut operations of subtrees in $O(\log n)$ amortised time. In addition, a link-cut tree allows finding the root of a node in $O(\log n)$ amortised time. The algorithm by Gudmundsson and Wong [24] used a link-cut tree to store and re-use monotone paths. Consider when a sweepline, either l_s or l_t, stops at a new critical point p. Instead of recomputing the monotone paths, they need only to add p to the existing link-cut tree they maintained in the previous instances.

With graph G defined, we can analyse the total running time of the algorithm by Gudmundsson and Wong [24] on the simplified free space diagram. The key to observe the running time is that in their algorithm, if an edge leads to a dead-end, it is marked and will not be used in future searches. Furthermore, inserting or removing an edge takes $O(\log n)$ amortised time in a link-cut tree.

▶ **Theorem 2.** *There is an $O(nm\log(c/\varepsilon)\log(n/\varepsilon))$ time algorithm that solves* $\texttt{SC}(\texttt{T}, \texttt{m}, \texttt{l}, (\texttt{1} + \varepsilon)\texttt{d})$ *in the case that the reference trajectory is vertex-to-vertex.*

Proof. Construction of the simplified free space diagram takes $O((cn/\varepsilon)\log(cn/\varepsilon))$ time by Theorem 1. Construction of G takes $O((cn/\varepsilon)\log(n/\varepsilon))$ time by Lemma 14. The graph G has at most $O(nm\log(c/\varepsilon))$ edges, see the full version. Gudmundsson and Wong [24] showed that

an edge is added to and removed from the link-cut tree at most once, and adding/removing an edge from the link-cut tree takes $O(\log(n/\varepsilon))$ time since the maximum number of nodes in the link-cut tree is upperbounded by the number of nodes in G. Therefore maintaining the link-cut tree takes $O(nm \log(c/\varepsilon) \log(n/\varepsilon))$ time. ◀

6 Reference trajectory is arbitrary

Our results in this section rely heavily on the work of Gudmundsson and Wong [24]. Due to space constraints, we can only highlight important parts of their algorithm, and the analysis of our improvements.

When the reference trajectory is arbitrary, a monotone path can start and finish at arbitrary positions in the non-empty cells. Therefore, in addition to the critical points in the free space diagram and the greedy critical points, Gudmundsson and Wong defined three new types of *internal critical points* [24, Definition 25]. An internal critical point must lie in the interior of a non-empty cell, and lie on the boundary of the free space. They made the following distinction (see Figure 4).

1. End-of-cell critical point: the leftmost and rightmost white point of a non-empty cell.
2. Propagated critical point: a point on the boundary of the free space that shares a y-coordinate with a critical point.
3. l-apart critical points: two points on the boundaries of free space that are a distance of l apart horizontally.

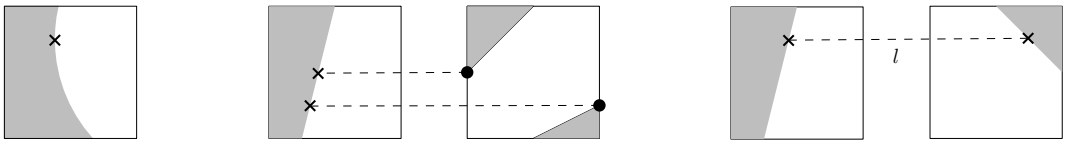


Figure 4 The three types of internal critical points are illustrated using a cross in the left, middle, and right figures, respectively. From left to right, they are the end-of-cell critical points (left), propagated critical points (middle) and l-apart critical points (right).

There could be an infinite number of l-apart critical points in a pair of non-empty cells. However, if this is the case, we can simply perturb the input by a miniscule amount so that there are no longer an infinite number of l-apart critical points. See the full version for an example and a figure. Therefore, for the rest of the paper, we can assume that there are at most a constant number of l-apart critical points per pair of cells.

We will first bound the number of internal critical points and the time it takes to compute them. One can compute the end-of-cell and l-apart critical points in linear time with respect to the number of non-empty cells since there are at most a constant number of them per pair of cells. In Lemma 13, we showed that in a narrow horizontal strip, only a small number of cells intersect free space. An output-sensitive query algorithm would be efficient to find the non-empty cells that a critical point p propagates to. Therefore, we can use an interval tree [14] to store the y-spans of all non-empty cells in a row, and query the intersecting intervals of $y(p)$ in logarithmic time. We formalise the above arguments in the below Lemma 15.

▶ **Lemma 15.** *Assume that there is a constant number of l-apart critical points per pair of cells, it takes $O(cn\log(n/\varepsilon) + c^2n/\varepsilon^2)$ time to compute $O(c^2n/\varepsilon^2)$ internal critical points in the simplified free space diagram $\mathcal{F}'_{(1+\hat{\varepsilon})d}(T,T)$.*

Proof. There are $O(cn/\varepsilon)$ non-empty aggregated cells in $\mathcal{F}'_{(1+\hat{\varepsilon})d}(T,T)$, or non-empty cells for short, and $O(cn/\varepsilon)$ end-of-cell critical points in $\mathcal{F}'_{(1+\hat{\varepsilon})d}(T,T)$. Each critical point propagates $O(c/\varepsilon)$ times by Lemma 13, therefore there are $O(c^2n/\varepsilon^2)$ propagated critical points. We can charge a cell with a constant number of l-apart critical points. Therefore, there are at most $O(cn/\varepsilon)$ l-apart critical points. In total, there are $O(c^2n/\varepsilon^2)$ internal critical points.

One can compute the end-of-cell critical points by iterating through the free space diagram in $O(cn/\varepsilon)$ time. To compute the l-apart critical points, we can start from the first non-empty cell C in a row and find the first cell that is l-apart from C, and solve a constant number of quadratic equations. We can then slide this l-apart line and do the same for all pairs of cells that are l-apart in all rows in $O(cn/\varepsilon)$ time in total.

To compute the propagated critical points, we construct an interval tree [14] for each row in $\mathcal{F}'_{(1+\hat{\varepsilon})d}(T,T)$ to store the maximum and minimum y-coordinates of the free space in the non-empty cells. Let n_i be the number of non-empty cells in the ith row. We can sum the construction time of the interval trees.

$$\sum_{i=1}^{n} n_i \log n_i \leq \frac{cn}{\varepsilon}\log\left(\frac{cn}{\varepsilon}\right) \in O\left(\left(\frac{cn}{\varepsilon}\right)\log\left(\frac{cn}{\varepsilon}\right)\right)$$

Given a critical point p in the ith row, one can query the interval tree in $O(\log n_i + c/\varepsilon) \in O(\log n + c/\varepsilon)$ time to compute the propagated critical points from p using Lemma 13 and [14]. With $O(cn/\varepsilon)$ critical points, computing the propagated critical points takes $O(cn\log(n)/\varepsilon + c^2n/\varepsilon^2)$ time. ◀

With the additional internal critical points, the number of reference trajectories and the number of greedy critical points increases. We can use the algorithm in the previous section, and obtain the following result.

▶ **Lemma 16.** *There is an $O((c^2mn/\varepsilon^2)\log(c/\varepsilon)\log(n/\varepsilon))$ time algorithm that solves $\mathtt{SC}(\mathtt{T},\mathtt{m},\mathtt{l},(\mathtt{1}+\varepsilon)\mathtt{d})$ in the case that the reference trajectory is arbitrary.*

Proof. See the full version. ◀

6.1 Improve Further with an Interval Management Data Structure

The bottleneck in the above Lemma 16 is operating the outgoing edges of the $O(c^2mn/\varepsilon^2)$ greedy critical points, which are generated from $O(c^2n/\varepsilon^2)$ propagated critical points. To avoid computing the greedy critical points, Gudmundsson and Wong [24] used a dynamic monotonic interval data structure [16] to store overlapping monotonic intervals that represent the y-spans of monotone paths between l_s and l_t. Instead of searching for a set of monotone paths between each window greedily, they showed that one can update and query the interval data structure to retrieve $m-1$ non-overlapping intervals, all in $O(\log n)$ amortised time.

▶ **Theorem 3.** *There is an $O((c^2n/\varepsilon^2)\log(c/\varepsilon)\log(n/\varepsilon))$ time algorithm that solves $\mathtt{SC}(\mathtt{T},\mathtt{m},\mathtt{l},(\mathtt{1}+\varepsilon)\mathtt{d})$ in the case that the reference trajectory is arbitrary.*

Proof. Constructing the simplified free space diagram takes $O((cn/\varepsilon)\log(cn/\varepsilon))$ time by Theorem 1. Computing and sorting the internal critical points takes $O((c^2n/\varepsilon^2)\log(n/\varepsilon))$ time by Lemma 15. There are $O((c^2n/\varepsilon^2)\log(c/\varepsilon))$ edges in total by Lemma 13, and each

edge takes $O(\log(n/\varepsilon))$ time to insert or remove since there are at most $O(c^2 n/\varepsilon^2)$ nodes in the link-cut tree. In total, we spend $O((c^2 n/\varepsilon^2)\log(c/\varepsilon)\log(n/\varepsilon))$ time to maintain the edges in G.

Each internal critical point is treated as an event, and maintaining the interval data structure takes $O(\log n)$ amortised time per event point (see [24, Theorem 2]), and thus $O((c^2 n/\varepsilon^2)\log n)$ in total. The overall complexity is dominated by maintaining the edges. ◀

7 Conclusion

We presented an algorithm that solves the subtrajectory cluster problem on c-packed trajectories T with an approximation error on the Fréchet distance, achieving an $O((c^2 n/\varepsilon^2)\log(c/\varepsilon)\log(n/\varepsilon))$ time complexity. Our algorithm builds upon the near-optimal algorithm proposed by Gudmundsson and Wong [24], but with significant improvements. By carefully analysing the properties of c-packed trajectories, we have shown that important parameters such as the number of propagated critical points are significantly lower than the theoretical $O(n)$ upperbound for realistic trajectories. As a result, our algorithm improves upon the near-optimal algorithm by replacing a factor of n^2 with c^2/ε^2, leading to more efficient subtrajectory cluster of realistic trajectories.

References

1 Pankaj K. Agarwal, Kyle Fox, Kamesh Munagala, Abhinandan Nath, Jiangwei Pan, and Erin Taylor. Subtrajectory Clustering: Models and Algorithms. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '18, pages 75–87, New York, NY, USA, May 2018. Association for Computing Machinery. doi:10.1145/3196959.3196972.

2 Sepideh Aghamolaei, Vahideh Keikha, Mohammad Ghodsi, and Ali Mohades. Windowing queries using Minkowski sum and their extension to MapReduce. *The Journal of Supercomputing*, 77(1):936–972, January 2021. doi:10.1007/s11227-020-03299-7.

3 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 05:75–91, March 1995. doi:10.1142/S0218195995000064.

4 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670, October 2014. ISSN: 0272-5428. doi:10.1109/FOCS.2014.76.

5 Karl Bringmann and Marvin Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *International Journal of Computational Geometry & Applications*, 27:85–119, March 2017. doi:10.1142/S0218195917600056.

6 Frederik Brüning, Jacobus Conradi, and Anne Driemel. Faster Approximate Covering of Subcurves Under the Fréchet Distance. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.ESA.2022.28.

7 Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristan, Rodrigo I. Silveira, Frank Staals, and Carola Wenk. Clustering trajectories for map construction. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '17, pages 1–10, New York, NY, USA, November 2017. Association for Computing Machinery. doi:10.1145/3139958.3139964.

8 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry & Applications*, 21(03):253–282, June 2011. doi:10.1142/S0218195911003652.

9 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, SODA '09, pages 645–654, USA, January 2009. Society for Industrial and Applied Mathematics.

10 Claudia Cavallaro, Armir Bujari, Luca Foschini, Giuseppe Di Modica, and Paolo Bellavista. Measuring the impact of COVID-19 restrictions on mobility: A real case study from Italy. *Journal of Communications and Networks*, 23(5):340–349, October 2021. doi:10.23919/JCN.2021.000034.

11 Cheng Chang and Baoyao Zhou. Multi-granularity visualization of trajectory clusters using sub-trajectory clustering. In *2009 IEEE International Conference on Data Mining Workshops*, pages 577–582, December 2009. doi:10.1109/ICDMW.2009.24.

12 Daniel Chen, Anne Driemel, Leonidas J. Guibas, Andy Nguyen, and Carola Wenk. Approximate map matching with respect to the Fréchet distance. In *2011 Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, Proceedings, pages 75–83. Society for Industrial and Applied Mathematics, January 2011. doi:10.1137/1.9781611972917.8.

13 Jacobus Conradi and Anne Driemel. On Computing the k-Shortcut Fréchet Distance. In Miko\laj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.ICALP.2022.46.

14 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-77974-2.

15 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, July 2012. doi:10.1007/s00454-012-9402-z.

16 Alexander Gavruskin, Bakhadyr Khoussainov, Mikhail Kokho, and Jiamou Liu. Dynamic algorithms for monotonic interval scheduling problem. *Theoretical Computer Science*, 562:227–242, January 2015. doi:10.1016/j.tcs.2014.09.046.

17 Joachim Gudmundsson, Martin P. Seybold, and John Pfeifer. On practical nearest sub-trajectory queries under the Fréchet distance. In *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '21, pages 596–605, New York, NY, USA, November 2021. Association for Computing Machinery. doi:10.1145/3474717.3484264.

18 Joachim Gudmundsson, Martin P. Seybold, and Sampson Wong. Map matching queries on realistic input graphs under the Fréchet distance. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1464–1492. Society for Industrial and Applied Mathematics, January 2023. doi:10.1137/1.9781611977554.ch53.

19 Joachim Gudmundsson, Yuan Sha, and Sampson Wong. Approximating the packedness of polygonal curves. *Computational Geometry*, 108:101920, January 2023. doi:10.1016/j.comgeo.2022.101920.

20 Joachim Gudmundsson and Michiel Smid. Fréchet queries in geometric trees. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms – ESA 2013*, pages 565–576, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-40450-4_48.

21 Joachim Gudmundsson, Andreas Thom, and Jan Vahrenhold. Of motifs and goals: mining trajectory data. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 129–138, New York, NY, USA, November 2012. Association for Computing Machinery. doi:10.1145/2424321.2424339.

22 Joachim Gudmundsson and Nacho Valladares. A GPU approach to subtrajectory clustering using the Fréchet distance. *IEEE Transactions on Parallel and Distributed Systems*, 26(4):924–937, April 2015. doi:10.1109/TPDS.2014.2317713.

23 Joachim Gudmundsson and Thomas Wolle. Football analysis using spatio-temporal tools. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 566–569, New York, NY, USA, November 2012. Association for Computing Machinery. doi:10.1145/2424321.2424417.

24 Joachim Gudmundsson and Sampson Wong. Cubic upper and lower bounds for subtrajectory clustering under the continuous Fréchet distance. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 173–189. Society for Industrial and Applied Mathematics, January 2022. doi:10.1137/1.9781611977073.9.

25 Amin Hosseinpoor Milaghardan, Rahim Ali Abbaspour, Christophe Claramunt, and Alireza Chehreghan. An activity-based framework for detecting human movement patterns in an urban environment. *Transactions in GIS*, 25(4):1825–1848, 2021. doi:10.1111/tgis.12749.

26 Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 593–604, New York, NY, USA, June 2007. Association for Computing Machinery. doi:10.1145/1247480.1247546.

27 Anil Maheshwari, Jörg-Rüdiger Sack, Kaveh Shahbaz, and Hamid Zarrabi-Zadeh. Fréchet distance with speed limits. *Computational Geometry*, 44(2):110–120, February 2011. doi:10.1016/j.comgeo.2010.09.008.

28 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, June 1983. doi:10.1016/0022-0000(83)90006-5.

29 Panagiotis Tampakis, Nikos Pelekis, Christos Doulkeridis, and Yannis Theodoridis. Scalable distributed subtrajectory clustering. In *Proceedings of the 2019 IEEE International Conference on Big Data (Big Data)*, pages 950–959. IEEE Computer Society, December 2019. doi:10.1109/BigData47090.2019.9005563.

30 Zheng Wang, Cheng Long, and Gao Cong. Similar sports play retrieval with deep reinforcement learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021. doi:10.1109/TKDE.2021.3136881.

Shortest Beer Path Queries in Digraphs with Bounded Treewidth

Joachim Gudmundsson
The University of Sydney, Australia

Yuan Sha
The University of Sydney, Australia

Abstract

A beer digraph G is a real-valued weighted directed graph where some of the vertices have beer stores. A beer path from a vertex u to a vertex v in G is a path in G from u to v that visits at least one beer store.

In this paper we consider the online shortest beer path query in beer digraphs with bounded treewidth t. Assume that a tree decomposition of treewidth t on a beer digraph with n vertices is given. We show that after $O(t^3 n)$ time preprocessing on the beer digraph, (i) a beer distance query can be answered in $O(t^3\alpha(n))$ time, where $\alpha(n)$ is the inverse Ackermann function, and (ii) a shortest beer path can be reported in $O(t^3\alpha(n)L)$ time, where L is the number of edges on the path. In the process we show an improved $O(t^3\alpha(n)L)$ time shortest path query algorithm, compared with the currently best $O(t^4\alpha(n)L)$ time algorithm [Chaudhuri & Zaroliagis, 2000].

We also consider queries in a dynamic setting where the weight of an edge in G can change over time. We show two data structures. Assume t is constant and let β be any constant in $(0, 1)$. The first data structure uses $O(n)$ preprocessing time, answers a beer distance query in $O(\alpha(n))$ time and reports a shortest beer path in $O(\alpha(n)L)$ time. It can be updated in $O(n^\beta)$ time after an edge weight change. The second data structure has $O(n)$ preprocessing time, answers a beer distance query in $O(\log n)$ time, reports a shortest beer path in $O(\log n + L)$ time, and can be updated in $O(\log n)$ time after an edge weight change.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Graph algorithms, Shortest Path, Data structures, Bounded treewidth

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.35

1 Introduction

A *beer digraph* is a real-valued weighted directed graph $G = (V(G), E(G))$ in which some of the vertices have beer stores. For any two vertices $u, v \in V(G)$, a *shortest beer path* is a shortest path from u to v that visits at least one beer store. The *beer distance* from u to v is the weight of a shortest beer path from u to v. In this paper, we consider the problem of shortest beer path and beer distance queries for beer digraphs in both static and dynamic settings. In the dynamic setting, the weight of an edge in the graph can change.

The shortest path and distance queries are fundamental graph problems. There are numerous works on the subject in the literature. Thorup and Zwick [22] used the term "distance oracle" to refer to a compact data structure that can efficiently answer distance query between any two vertices. Ideally one would like a distance oracle with linear preprocessing time and space, and constant query time. However, it is well known that there are graphs for which no distance oracle with $o(n^2)$ bits of space and $O(1)$ query time exists. Because of this, researchers have focused their attention on restricted classes of graphs.

There has been extensive research on the class of planar graphs [5, 8, 9, 10, 11, 13, 18, 19]. We briefly highlight some of the most recent results for planar graphs. In [5], Charalampopoulos et al. showed a distance oracle with space $O(n^{1+\epsilon})$ and polylogarithmic query-time for any constant $\epsilon > 0$. Long and Pettie [18] showed (i) an oracle with space

$O(n \log^{2+o(1)} n)$ and query-time $O(n^{\epsilon})$ for any constant $\epsilon > 0$, and (ii) an oracle with space $O(n^{1+o(1)})$ and query-time $n^{o(1)}$. For a comparison of the existing distance oracles for planar graphs, refer to Table 1 in [18].

For the class of graphs with bounded treewidth, the best shortest path and distance query structure is attributed to Chaudhuri and Zaroliagis [6]. Let t be the treewidth of the input graph G and assume that a tree decomposition of G with treewidth t is given. They showed that after $O(t^3 n)$ preprocessing, distance queries can be answered in time $O(t^3 \alpha(n))$, and shortest path queries can be answered in time $O(t^4 \alpha(n) L)$, where L is the number of edges on the path and $\alpha(n)$ is the inverse Ackermann function.

Not surprisingly, answering shortest path or distance queries efficiently in a dynamic setting where the graph undergoes changes is hard. Roditty and Zwick [21] showed that the innocent looking decremental (only edge deletions) or incremental (only edge insertions) *single-source* shortest path (SSSP) problem is, in a strong sense, as hard as APSP (all-pairs shortest path). It is conjectured that APSP has no truly subcubic time ($O(n^{3-\epsilon})$) solution [23].

The shortest beer path problem is a generalization of the shortest path problem. Bacic et al. [2] considered the shortest beer path and beer distance queries for undirected outerplanar graphs with non-negative edge weights. They showed that after $O(n)$ time preprocessing a beer distance query can be answered in $O(\alpha(n))$ time, and a shortest beer path query can be answered in $O(L)$ time. Hanaka et al. [15] considered the shortest beer path and beer distance queries for graphs with bounded triconnected component size (which they extends to graphs with bounded treewidth). In this paper we study shortest beer path and beer distance queries for digraphs with bounded treewidth.

We first show an improved $O(t^3 \alpha(n) L)$ time shortest path query algorithm, compared with the $O(t^4 \alpha(n) L)$ time algorithm in [6]. Pettie [20] proved that for the *online MST (minimum spanning tree) verification* problem, answering m queries requires $\Omega(m\alpha(m, n))$ time. Bacic et al. [2] reduced the online MST verification problem to the beer distance query problem on trees, which implies that answering m beer distance queries on beer trees requires $\Omega(m\alpha(m, n))$ time. Thus our result of beer distance query in Table 1 is optimal for graphs with constant treewidth. Note that our shortest beer path and beer distance query structure answers shortest beer path queries in the same asymptotic time as answering shortest path queries, and answers beer distance queries in the same asymptotic time as answering distance queries (see Table 1).

We also consider the shortest beer path and beer distance queries in the dynamic setting where edge weights can change over time. Edge deletion (setting weight to infinity) and edge re-insertion are special cases of edge weight change. We do not consider general edge insertion, since our approach is based on tree decompositions of graphs while general edge insertion can greatly change the treewidths and break the tree decompositions. For this dynamic setting we give two dynamic shortest beer path query structures. See Table 2 for a comparison of the structures.

2 Preliminaries

Let $G = (V(G), E(G))$ be a weighted beer digraph with n vertices where some of the vertices have a beer store. The edge weights are specified by a weight function $wt : E(G) \longrightarrow \mathbb{R}$. Let $wt(u, v)$ be the weight of edge $\langle u, v \rangle$. The weight of a path in G is the sum of the weights of the edges on the path. For any two vertices $u, v \in E(G)$, a shortest path in G from u to v is a path from u to v with the minimum weight and is denoted as $\pi_G(u, v)$. The distance from

Table 1 Comparison of shortest path query structures. In the table t is the treewidth, L is the number of edges on the path, and $\alpha(n)$ is the inverse Ackermann function. Assume a tree decomposition of treewidth t is given for a bounded treewidth graph. Entries in boldface are new.

Source	Graph	Preproc.	Distance	Shortest path	Beer dist.	Beer path
[6]	directed, bounded treewidth	$O(t^3n)$	$O(t^3\alpha(n))$	$O(t^4\alpha(n)L)$	–	–
[2]	undirected, outer-planar	$O(n)$	$O(\alpha(n))$	$O(L)$	$O(\alpha(n))$	$O(L)$
Theorem 9	directed, bounded treewidth	$O(t^3n)$	$O(t^3\alpha(n))$	$\mathbf{O(t^3\alpha(n)L)}$	$\mathbf{O(t^3\alpha(n))}$	$\mathbf{O(t^3\alpha(n)L)}$

Table 2 Comparison of dynamic shortest path query structures on directed graphs with constant treewidth. β is any constant in $(0, 1)$, and L is the number of edges on the path. Entries in boldface are new.

Source	Preproc.	Distance	Shortest path	Beer dist.	Beer path	Edge weight update
[6]	$O(n)$	$O(\alpha(n))$	$O(L\alpha(n))$	–	–	$O(n^\beta)$
Theorem 13	$O(n)$	$O(\alpha(n))$	$O(L\alpha(n))$	$\mathbf{O(\alpha(n))}$	$\mathbf{O(L\alpha(n))}$	$\mathbf{O(n^\beta)}$
Theorem 23	$\mathbf{O(n)}$	$\mathbf{O(\log n)}$	$\mathbf{O(\log n + L)}$	$\mathbf{O(\log n)}$	$\mathbf{O(\log n + L)}$	$\mathbf{O(\log n)}$

u to v in G, denoted as $\delta_G(u, v)$, is the weight of $\pi_G(u, v)$. A shortest *beer path* in G from u to v is denoted as $\pi_{BG}(u, v)$ and the *beer distance* from u to v in G, denoted as $\delta_{BG}(u, v)$, is the weight of $\pi_{BG}(u, v)$.

Let V_1, V_2 and U be disjoint subsets of $V(G)$. U is a separator for V_1 and V_2 if every path in G from a vertex in V_1 (V_2) to a vertex in V_2 (V_1) goes through a vertex in U.

Let X be a subset of $V(G)$ and let $G[X]$ be the subgraph of G induced by X.

A *tree decomposition* of a graph G (directed or undirected) is a pair (X, T) in which $T = (I = V(T), E(T))$ is a tree and $X = \{X_i | i \in I\}$ is a family of subsets of $V(G)$ such that:

1. $\bigcup_{i\in I} X_i = V(G)$;
2. for each edge $e = (u, v) \in E(G)$ there exists a node $i \in I$ such that both u and v belong to X_i; and
3. for all $v \in V$, the set of nodes $\{i \in I | v \in X_i\}$ forms a connected subtree of T.

The set X_i is called the *bag of* node i. The *treewidth* of a tree decomposition is $\max_{i\in I} |X_i| - 1$. The treewidth of G is the minimum treewidth over all possible tree decompositions of G.

▶ **Theorem 1** ([4]). *For every constant $t \in N$, there exists an $O(n)$ time algorithm which tests whether a given n-vertex graph G has treewidth at most t and if so, outputs a tree-decomposition (X, T) of G with treewidth at most t, where $|V(T)| = n - t$. In $O(n)$ time, the tree decomposition (X, T) can be converted into a binary tree decomposition of $2(n - t)$ nodes and treewidth t.*

For a tree decomposition (X, T) of G, every edge $e = (i, j) \in E(T)$ corresponds to a separator of G. Let T_1 and T_2 be the subtrees of T obtained by removing e from T, then $X_i \cap X_j$ separates $\bigcup_{m\in V(T_1)} X_m - (X_i \cap X_j)$ from $\bigcup_{m\in V(T_2)} X_m - (X_i \cap X_j)$.

3 Shortest path and distance queries

The most efficient algorithms for shortest path and distance queries on digraphs with bounded treewidth to date are given by Chaudhuri and Zaroliagis [6] (see Table 1). We give some preliminaries in Section 3.1, before giving an outline of the algorithms in [6] in Section 3.2. Finally we show how to improve the shortest path query algorithm in Section 3.3.

3.1 Preliminaries

Call (a, b, c) a *distance tuple* if a, b are symbols and $c \in \mathbb{R}$. Define a composition operator $\circ$ on two distance tuples as $(a_1, b_1, c_1) \circ (a_2, b_2, c_2) = (a_1, b_2, c_1 + c_2)$ if $b_1 = a_2$, otherwise it is undefined. For a set of distance tuples, say M, define $minmap(M)$ to be the operation on M such that among all distance tuples in M with the same first and second components, only the distance tuple with the smallest third component is retained.

Let A and B be two sets of distance tuples. Define a composition operator $\circ$ on A and B as $A \circ B = minmap\{x \circ y | x \in A, y \in B\}$. For a node $i \in V(T)$ where (X, T) is a tree decomposition of G, define $\gamma(i) = \{(u, v, \delta_G(u, v)) | u, v \in X_i\}$ to be the set of distance tuples for pairs of vertices in X_i (u and v can be the same vertex).

▶ **Definition 2** ([6]). *Let H be a digraph, with V_1, V_2 and U be a partition of $V(H)$ such that U is a separator for V_1 and V_2. Let H_1 and H_2 be subgraphs of H such that $V(H_1) = V_1 \cup U$ and $V(H_2) = V_2 \cup U$. We say that H_1' is a graph obtained by absorbing H_2 into H_1, if H_1' is obtained from H_1 by adding edges $\langle u, v \rangle$, with weight $\delta_{H_2}(u, v)$ or $\delta_H(u, v)$, for each pair $u, v \in U$. In case multiple edges are created, retain the one with minimum weight.*

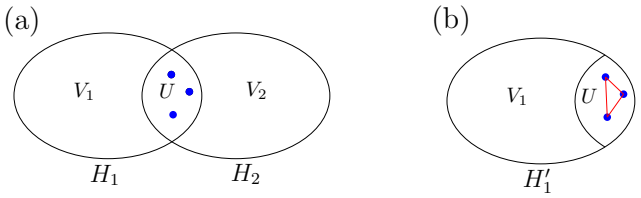


Figure 1 (a) Illustrating Definition 2. (b) H_1' where the red edges have weights $\delta_{H_2}(\cdot, \cdot)$ or $\delta_H(\cdot, \cdot)$.

Absorption preserves distances in a digraph. Let H, H_1, H_2 and H_1' be as in Definition 2, then for all $x, y \in V(H_1')$, $\delta_{H_1'}(x, y) = \delta_H(x, y)$.

3.2 Shortest path and distance query algorithms

Let G be a weighted digraph with bounded treewidth t. Here we will briefly present the key ideas of the algorithms by Chandhuri and Zaroliagis [6]. From Theorem 1, one can obtain a tree decomposition (X, T) of G with constant treewidth in $O(n)$ time.[1] One can obtain $\gamma(i)$ for all nodes $i \in V(T)$ in $O(t^3 n)$ time by the following lemma. The algorithm uses an *absorption and expansion processes*. $Int_G(u, v)$ is an intermediate vertex (neither u nor v) on a shortest path from u to v.

[1] The hidden constant of the $O(n)$ time in Theorem 1 is $2^{O(t^3)}$. Recently, Korhonen [17] gave a 2-approximation algorithm that runs in $O(2^{O(t)} n)$ time, which suffices for our application. Interested readers are also referred to [12] for an $O(t^7 n \log n)$ time, $O(t)$-approximation algorithm.

▶ **Lemma 3.** *Let G be an n-vertex weighted digraph and let (X, T) be the tree decomposition of G, of treewidth t. For each pair u, v such that $u, v \in X_i$ for some $i \in V(T)$, let $Dist(u, v) = \delta_G(u, v)$ and $Int_G(u, v) = x$, where x is some intermediate vertex (neither u nor v) on a shortest path from u to v. If $wt(u, v) = \delta_G(u, v)$, then $Int_G(u, v) = null$. Then in $O(t^3 n)$ time, we can either find a negative cycle in G, or compute the values $Dist(u, v)$ and $Int_G(u, v)$ for each such pair u, v.*

After using the algorithm in Lemma 3, all $\gamma(i)$ for $i \in V(T)$ have been computed. One can define a semigroup $(\Gamma, \circ)$ where Γ is the set of sets of distance tuples and $\circ$ is the composition operator defined on two sets of distance tuples. Label node $i \in V(T)$ with $\gamma(i)$. For any two node $i, j \in V(T)$, the composition of the labels along the path in T from i to j, which is $\gamma(i) \circ \ldots \circ \gamma(j)$, gives the set of distance tuples $P(X_i, X_j) = \{(a, b, \delta_G(a, b)) | a \in X_i, b \in X_j\}$. This is true because each edge along the path from i to j corresponds to a separator of G (see the text below Theorem 1. Thus for any two vertices $u, v \in V(G)$, if $u \in X_i$ and $v \in X_j$, then the composition of the labels along the path in T from i to j gives $\delta_G(u, v)$. The following theorem is proved in [1] and [7].

▶ **Theorem 4** ([1, 7])**.** *Let $(S, \circ)$ be a semigroup such that for $q, r \in S$, $q \circ r$ can be computed in $O(m)$ time. Let T be a tree with n nodes where each node is labeled with an element from the semigroup. After $O(mn)$ time preprocessing, the composition of the labels along any path in T can be computed in $O(m\alpha(n))$ time.*

Plug in the result in Theorem 4, after $O(t^3 n)$ time preprocessing, the composition of the labels along any path in T of (X, T) can be computed in $O(t^3\alpha(n))$ time. Therefore the distance between any two vertices in $V(G)$ can be computed in $O(t^3\alpha(n))$ time.

The shortest path query algorithm uses the distance query algorithm. It works recursively and finds one intermediate vertex on the shortest path at a time. In a recursive step, the algorithm checks t vertices and for each vertex it makes two distance queries. Since a distance query is answered in $O(t^3\alpha(n))$ time, an intermediate vertex is found in $O(t^4\alpha(n))$ time. The shortest path is reported in $O(t^4\alpha(n)L)$ time, where L is the number of edges on the path. In summary:

▶ **Theorem 5** ([6])**.** *Let G be an n-vertex weighted digraph of treewidth t and assume a tree decomposition of G with treewidth t is given. After $O(t^3 n)$ time preprocessing, distance queries in G can be answered in time $O(t^3\alpha(n))$, and shortest path queries in G can be answered in time $O(t^4\alpha(n)L)$, where L is the number of edges on the path.*

3.3 Our improved shortest path query algorithm

In this section we show how to improve the shortest path query algorithm. The shortest path query algorithm in Section 3.2 computes an intermediate vertex by checking t vertices and making $O(t)$ distance queries. However, we can define *augmented* distance tuples which contain intermediate vertex information, and define the composition operator $\circ$ on two augmented distance tuples such that the composition not only gives distance but also gives intermediate vertex. Using the augmented distance tuples and the composition operator on them, we can compute an intermediate vertex by making just one distance query.

An augmented distance tuple (a, b, c, d) has a fourth component d which is the intermediate vertex information. The composition operator $\circ$ on two augmented distance tuples is defined as $(a_1, b_1, c_1, d_1) \circ (a_2, b_2, c_2, d_2) = (a_1, b_2, c_1 + c_2, d')$ if $b_1 = a_2$, otherwise it is undefined. When $b_1 = a_2$, the fourth component d' is determined as follows. If $a_1 = b_1$ or $a_2 = b_2$, then $d' = d_2$ or $d' = d_1$, respectively. Otherwise, $a_1 \neq b_1$ and $a_2 \neq b_2$. Then if $a_1 \neq b_2$,$d' = b_1$. If $a_1 = b_2$, $d' = null$.

Now we redefine the semigroup $(\Gamma, \circ)$. An element in the redefined Γ is a set of augmented distance tuples. Let A and B be two sets of augmented distance tuples. The composition operator $\circ$ on A and B is defined as $A \circ B = minmap\{x \circ y | x \in A, y \in B\}$, where the *minmap* operation is as defined before. For a node $i \in V(T)$, let $\bar{\gamma}(i) = \{(u, v, \delta_G(u, v), Int_G(u, v)) | u, v \in X_i\}$. The values $\delta_G(u, v)$ and $Int_G(u, v)$ have been computed by Lemma 3. Relabel node $i \in V(T)$ with $\bar{\gamma}(i)$. The labels are elements in the redefined $(\Gamma, \circ)$. For any two nodes $i, j \in V(T)$, the composition of the labels along the path in T from i to j, $\bar{\gamma}(i) \circ \ldots \circ \bar{\gamma}(j)$, gives the set of augmented distance tuples $\bar{P}(X_i, X_j) = \{(a, b, \delta_G(a, b), Int_G(a, b)) | a \in X_i, b \in X_j\}$. For any two vertices $u, v \in V(G)$, if $u \in X_i$ and $v \in X_j$, then the composition of the labels along the path in T from i to j gives $\delta_G(u, v)$ and $Int_G(u, v)$.

The composition of the labels along a path in T gives the claimed set of augmented distance tuples, since each edge along the path corresponds to a separator of G.

For two labels $\bar{\gamma}(i)$ and $\bar{\gamma}(j)$, one can compute $\bar{\gamma}(i) \circ \bar{\gamma}(j)$ in $O(t^3)$ time. If we plug in the data structure in Theorem 4, then after $O(t^3 n)$ time preprocessing, the composition of the labels $\bar{\gamma}(\cdot)$ along any path in T can be computed in $O(t^3 \alpha(n))$ time. We have the following theorem.

▶ **Theorem 6.** *Let G be an n-vertex weighted digraph of treewidth t and assume a tree decomposition of G with treewidth t is given. After $O(t^3 n)$ time preprocessing, distance queries in G can be answered in time $O(t^3 \alpha(n))$, and shortest path queries in G can be answered in time $O(t^3 \alpha(n) L)$, where L is the number of edges on the path.*

4 Shortest beer path and beer distance queries

Throughout this section, let G be a beer digraph. In this section we extend the approach discussed in Section 3 to shortest beer path and beer distance queries. Given a tree decomposition (X, T) of the input digraph G, we apply the *absorption and expansion* processes similar to those in Section 3 on (X, T), which compute information on beer paths and beer distances. Then we define a semigroup whose elements are sets of augmented distance tuples and augmented beer distance tuples. Finally we use the data structure in Theorem 4 to answer shortest beer path and beer distance queries efficiently. Before describing the algorithms, we give some preliminaries.

4.1 Preliminaries

Let H be a beer digraph. We call a vertex $v \in V(H)$ a beer vertex if v has a beer store. Let $u, v \in V(H)$. Recall that we use $\pi_{BH}(u, v)$ to denote a shortest beer path in H from u to v and use $\delta_{BH}(u, v)$ to denote the the weight of $\pi_{BH}(u, v)$. Let $Int_{BH}(u, v)$ denote an intermediate vertex (neither u nor v) on $\pi_{BH}(u, v)$. If $wt(u, v) = \delta_{BH}(u, v)$, then $Int_{BH}(u, v) = null$. If either u or v is a beer vertex, a shortest path from u to v is a shortest beer path from u to v and we define $Int_{BH}(u, v)$ to be an intermediate vertex (neither u nor v) on $\pi_H(u, v)$. If neither u nor v is a beer vertex, we define $Int_{BH}(u, v)$ to be a beer vertex that is on $\pi_{BH}(u, v)$.

Corresponding to a shortest beer path $\pi_{BH}(u, v)$, we define a beer edge $\langle u, v \rangle_B$ which has weight $\delta_{BH}(u, v)$ and the intermediate vertex $Int_{BH}(u, v)$. Note that a beer edge is different from a normal edge. We extend Definition 2 to beer graphs.

▶ **Definition 7.** *Let H be a beer digraph, possibly with beer edges. Let V_1, V_2 and U be a partition of $V(H)$ such that U is a separator for V_1 and V_2. Let H_1 and H_2 be subgraphs of H such that $V(H_1) = V_1 \cup U$ and $V(H_2) = V_2 \cup U$. We say that H_1' is a beer graph*

obtained by absorbing H_2 into H_1, if H_1' is obtained from H_1 by adding edges $\langle u,v\rangle$, with weight $\delta_{H_2}(u,v)$ or $\delta_H(u,v)$, and beer edges $\langle u,v\rangle_B$, with weight $\delta_{BH_2}(u,v)$ or $\delta_{BH}(u,v)$, for each pair $u,v \in U$. In case multiple edges or multiple beer edges are created, retain the one with minimum weight.

Absorption preserves distances and beer distances in a beer digraph. Let H, H_1, H_2 and H_1' be as in Definition 7, then for all $x,y \in V(H_1')$, $\delta_{H_1'}(x,y) = \delta_H(x,y)$ and $\delta_{BH_1'}(x,y) = \delta_{BH}(x,y)$.

For a node $i \in V(T)$ where (X,T) is a tree decomposition of G, define $\bar{\gamma}_B(i) = \{(u,v,\delta_G(u,v), Int_G(u,v))|u,v \in X_i\} \cup \{(u,v,\delta_{BG}(u,v), Int_{BG}(u,v))|u,v \in X_i\}$ to be the set of augmented distance tuples and augmented beer distance tuples for pairs of vertices in X_i.

Given a tree decomposition (X,T) of treewidth t of G, we use the absorption and expansion processes with the absorbing procedure defined in Definition 7, to compute $\bar{\gamma}(i)$ and $\bar{\gamma}_B(i)$ for all $i \in V(T)$. We have the following lemma.

▶ **Lemma 8.** *Let G be an n-vertex weighted digraph and let (X,T) be a tree decomposition of G, of treewidth t. Then in $O(t^3 n)$ time, we can compute the values $\delta_G(u,v)$, $Int_G(u,v)$, $\delta_{BG}(u,v)$ and $Int_{BG}(u,v)$ for each pair $u,v \in X_i$ for every $i \in V(T)$.*

4.2 Defining a semigroup

We first define a composition operator $\circ_B$ on augmented distance tuples and augmented beer distance tuples. When both operands are augmented distance tuples, the operator $\circ_B$ equals the operator $\circ$ defined in Section 3.1. The operator $\circ_B$ is undefined when both operands are augmented beer distance tuples. It remains to define $\circ_B$ between an augmented distance tuple and an augmented beer distance tuple. Let (a_1,b_1,c_1,d_1) be an augmented distance tuple and let $(a_2,b_2,\hat{c}_2,\hat{d}_2)$ be an augmented beer distance tuple. We only show the definition of $(a_1,b_1,c_1,d_1) \circ_B (a_2,b_2,\hat{c}_2,\hat{d}_2)$, since the definition of $(a_2,b_2,\hat{c}_2,\hat{d}_2) \circ_B (a_1,b_1,c_1,d_1)$ is symmetric. Define $(a_1,b_1,c_1,d_1) \circ_B (a_2,b_2,\hat{c}_2,\hat{d}_2) = (a_1,b_2,c_1+\hat{c}_2,\hat{d})$ if $b_1 = a_2$, otherwise it is undefined. The tuple $(a_1,b_2,c_1+\hat{c}_2,\hat{d})$ is a beer distance tuple. The fourth component $\hat{d}$ is set as follows. Here we assume that a_1, b_1 and b_2 are all different. The other cases are very similar. If a_1 or b_1 is a beer vertex, then $\hat{d} = b_1$. Otherwise neither a_1 nor b_1 is a beer vertex. Then, if a_2 or b_2 is a beer vertex, $\hat{d} = a_2$, otherwise $\hat{d} = \hat{d}_2$. It is not hard to verify that the setting of $\hat{d}$ is consistent with the definition of an augmented beer distance tuple. The setting of $\hat{d}$ takes constant time in all cases, therefore the composition $(a_1,b_1,c_1,d_1) \circ_B (a_2,b_2,\hat{c}_2,\hat{d}_2)$ takes constant time.

Let $\hat{A}$ and $\hat{B}$ be two sets of augmented distance tuples and augmented beer distance tuples. That is, $\hat{A} = A_1 \cup A_2$ and $\hat{B} = B_1 \cup B_2$ where A_1 (B_1) is a set of augmented distance tuples and A_2 (B_2) is a set of augmented beer distance tuples. Define the composition operator $\circ_B$ on $\hat{A}$ and $\hat{B}$ as $\hat{A} \circ_B \hat{B} = minmap\{x \circ_B y|x \in \hat{A}, y \in \hat{B}\}$, where the *minmap* is the operation such that among all distance tuples (or all beer distance tuples) with the same first and second components, only the distance tuple (or the beer distance tuple) with the smallest third component is retained. The operation $\circ_B$ is associative.

Recall that

$$\bar{\gamma}_B(i) = \{(u,v,\delta_G(u,v), Int_G(u,v))|u,v \in X_i\} \cup \{(u,v,\delta_{BG}(u,v), Int_{BG}(u,v))|u,v \in X_i\}$$

for all $i \in V(T)$ can be computed in $O(t^3 n)$ time according to Lemma 8. We define a semigroup $(\Gamma_B, \circ_B)$ where Γ_B is the set of sets of augmented distance tuples and augmented

beer distance tuples, $\circ_B$ is the composition operator defined on two elements in Γ_B. Label each node $i \in V(T)$ with $\bar{\gamma}_B(i)$. Since the composition $\circ_B$ of two tuples takes constant time, the composition of two labels takes $O(t^3)$ time. For any two nodes $i, j \in V(T)$, the composition of the labels along the path in T from i to j gives the set $\{(a, b, \delta_G(a, b), Int_G(a, b))|a \in X_i, b \in X_j\} \cup \{(a, b, \delta_{BG}(a, b), Int_{BG}(a, b))|a \in X_i, b \in X_j\}$.

Plug in Theorem 4, we have that after $O(t^3 n)$ time preprocessing, the composition of the labels along any path in T of (X, T) can be computed in $O(t^3\alpha(n))$ time. Thus the beer distance between any two vertices $u, v \in V(G)$ and the intermediate vertex $Int_{BG}(u, v)$ can be computed in $O(t^3\alpha(n))$ time.

The shortest beer path query algorithm is straightforward. If either u or v is a beer vertex, then we can use the shortest path query algorithm to compute the shortest beer path from u to v in $O(t^3\alpha(n)L)$ time by Theorem 6, where L is the number of edges on the beer path. If neither u nor v is a beer vertex, we first make a beer distance query from u to v. The beer distance query returns the intermediate vertex $Int_{BG}(u, v)$, which is a beer vertex. Then we use the shortest path query algorithm to compute the shortest beer path from u to $Int_{BG}(u, v)$ and the shortest beer path from $Int_{BG}(u, v)$ to v. The concatenation of the two paths is a shortest beer path from u to v, which is computed in $O(t^3\alpha(n)L)$ time. We have obtained the following theorem.

▶ **Theorem 9.** *Let G be an n-vertex beer digraph of treewidth t and assume a tree decomposition of G with treewidth t is given. After $O(t^3 n)$ time preprocessing on G, beer distance queries can be answered in time $O(t^3\alpha(n))$, and shortest beer path queries can be answered in time $O(t^3\alpha(n)L)$, where L is the number of edges on the path.*

5 A dynamic shortest beer path query structure

In this section we give a dynamic shortest beer path query structure, which is an extension of the dynamic shortest path query structure in [6]. We give an outline of the dynamic shortest path query structure in the following. The dynamic shortest beer path query structure is shown in Section 5.2.

The main technical tool is a graph partitioning algorithm. Let (X, T) be a *binary* tree decomposition of G of treewidth t. One can convert a tree decomposition of treewidth t into a binary tree decomposition of treewidth t in $O(tn)$ time. Given any integer m, the graph partitioning algorithm partitions the tree T of (X, T) in $O(n)$ time into $q \leq 16n/m$ node-disjoint subtrees $\{T_i|1 \leq i \leq q\}$ such that T_i has at most m nodes and is connected to the rest of T via at most three nodes. For each T_i, a subgraph H_i which is the subgraph of G induced by vertices in $\bigcup_{v \in V(T_i)} X_v$ is created. The subgraph H_i has T_i as its tree decomposition. A reduced graph H' is also created. Let v_1, v_2 and v_3 be the nodes of subtree T_i via which T_i is connected to the rest of T. The set $C(H_i) = X_{v_1} \cup X_{v_2} \cup X_{v_3}$ is called the *cut set* of H_i and contains at most $3t$ vertices. By shrinking each subtree T_i into a node with $C(H_i)$ as its bag of vertices, one creates a reduced tree T' with $q \leq 16n/m$ nodes. The reduced graph H' has T' as its tree decomposition and includes edges in G joining pairs of vertices in $C(H_i)$, $1 \leq i \leq q$.

The input graph G is partitioned into edge-disjoint components $\{G_i|1 \leq i \leq q\}$ where G_i is H_i with edges joining pairs of vertices in $C(H_i)$ deleted. Construct graph G' from H' by adding edges $\langle x, y\rangle$ weighted $\delta_{G_i}(x, y)$ for each pair x, y in $C(H_i)$, $1 \leq i \leq q$. Multiple edges in G' are replaced by the edge of minimum weight. Note that G' is the graph obtained by absorbing $G_1, \ldots, G_q$ into the rest of G. It follows that $\delta_{G'}(u, v) = \delta_G(u, v)$, for any $u, v \in V(G')$. By setting $m = 8\sqrt{n}$, H_i has at most $8t\sqrt{n}$ vertices and H' has at most $3tq \leq 6t\sqrt{n}$ vertices.

When the shortest path/distance query structures have been built for G' and G_i, $1 \leq i \leq q$, we can compute the shortest path/distance between any two vertices $u, v \in V(G)$ by querying the structures. If $u \in V(G_i)$ and $v \in V(G_j) \setminus V(G_i)$ for some i and j, we have

$$\delta_G(u,v) = \min\{\delta_{G_i}(u,x) + \delta_{G'}(x,y) + \delta_{G_j}(y,v) | x \in C(G_i), y \in C(G_j)\}. \quad (1)$$

If $u, v \in V(G_i)$ for some i, we have

$$\delta_G(u,v) = \min\{\delta_{G_i}(u,v), \min\{\delta_{G_i}(u,x) + \delta_{G'}(x,y) + \delta_{G_i}(y,v) | x, y \in C(G_i)\}\}. \quad (2)$$

Replacing the distances realizing $\delta_G(u,v)$ in Equation (1) or Equation (2) by the corresponding shortest paths gives the shortest path from u to v.

An edge in $E(G)$ corresponds to an edge in exactly one graph in $\{G_i | 1 \leq i \leq q\} \cup G'$. If the weight of the edge is updated and the edge is in G', one only needs to update the structure for G'. If the edge is in G_i, one needs to update the structure for G_i. Since an edge $\langle x, y \rangle$ with weight $\delta_{G_i}(x,y)$ was added in G' for each pair $x, y \in C(H_i)$, the change of an edge weight in G_i can incur at most $(3t)^2$ edge weight changes in G'. Thus one needs to update the structure for G' for these edge weight updates. To make an edge weight update efficient, the above procedure of graph partitioning and construction of a reduced graph is repeated *recursively* for each of $G_1, \ldots, G_q, G'$ until the component subgraphs at the deepest recursion level have small sizes. A *static* query structure in Section 3 is built for each component subgraph at the deepest recursion level. The dynamic data structure can be thought of as a tree structure where the root is G and every other node is $\bar{G}_i$ or $\bar{G}'$ of its parent $\bar{G}$. A static query structure is built and associated with each leaf node.

Since the dynamic data structure is built recursively, distance query or shortest path query is answered by recursively querying lower recursion level structures. Eventually queries are made and answered at the static query structures at the bottom level. An edge weight update is accommodated by recursively updating lower level structures. Eventually an update at a static query structure at the bottom level is accommodated by rebuilding the static query structure.

Let $r-1$ denote the number of recursion steps. The dynamic shortest path query structure is summarized in the following theorem.

▶ **Theorem 10** ([6]). *Let G be an n-vertex weighted digraph and assume a binary tree decomposition of G with treewidth t is given. For any positive integer constant r, one can build a data structure in $O(C_r t^{r+3} n)$ time such that the structure answers distance queries in $O(C_r t^{2r+2} \alpha(n))$ time, answers shortest path queries in $O(C_r t^{2r+2} \alpha(n) L)$ time where L is the number of edges on the shortest path, and accommodates an edge weight update in $O(C_r t^{2r+2} n^{(1/2)^{r-1}})$ time, where $C_r = 3^{r(r+2)}$.*

5.1 On the shortest path query time

We observe that the shortest path query algorithm only needs to compute the shortest path from u to v after making all the distance queries used to determine the shortest path from u to v. The shortest path from u to v is computed by making shortest path queries to static query structures at the bottom level and concatenating the computed shortest subpaths together, where a shortest subpath is a subpath of the shortest path from u to v. A shortest path query to a static query structure is answered in $O((3^{r-1}t)^4 \alpha(n) L_i)$ time, where $3^{r-1}t$ is the maximum treewidth of a graph at the bottom recursion level and L_i is the number of edges on the computed shortest subpath. Therefore the shortest path from u to v can be computed in $O(\sum_i (3^{r-1}t)^4 \alpha(n) L_i) = O(3^r t^4 \alpha(n) L)$ time where L is the number of edges

on the shortest path from u to v, besides the $O(C_r t^{2r+2}\alpha(n))$ time spent on making all the distance queries. If we use the shortest path query algorithm in Theorem 6 of Section 3.3, the $O(3^r t^4 \alpha(n)L)$ time is improved to $O(3^r t^3 \alpha(n)L)$ time. We have the following corollary.

▶ **Corollary 11.** *With the static shortest path query structure in Theorem 6, the shortest path query time in Theorem 10 can be improved to* $O(C_r t^{2r+2}\alpha(n) + 3^r t^3 \alpha(n)L)$.

5.2 Dynamic shortest beer path and beer distance queries

Equipped with the graph partitioning technique and the shortest beer path query structure in Theorem 9, we are ready to devise a dynamic shortest beer path and beer distance query structure, which is an extension of the dynamic shortest path query structure in Theorem 10. In the following we only focus on the differences.

We use the graph partitioning algorithm to partition the input beer graph G into edge-disjoint components $\{G_i | 1 \leq i \leq q\}$ where $q \leq 2\sqrt{n}$, the same as was done by the dynamic shortest path query structure. The reduced graph H' is the same as in the dynamic shortest path query structure. A reduced beer graph G' is constructed from H' by adding edges $\langle x, y\rangle$ weighted $\delta_{G_i}(x, y)$ and beer edges $\langle x, y\rangle_B$ weighted $\delta_{BG_i}(x, y)$ for each pair $x, y \in C(H_i)$, $1 \leq i \leq q$. Edges and beer edges are dealt with separately. Multiple edges in G' are replaced by the edge of minimum weight, and multiple beer edges in G' are replaced by the beer edge of minimum weight. The beer graph G' is obtained by absorbing $G_1, \ldots, G_q$ into the rest of G using the absorption defined in Definition 7. It follows that $\delta_{G'}(u, v) = \delta_G(u, v)$ and $\delta_{BG'}(u, v) = \delta_{BG}(u, v)$, for any $u, v \in V(G')$.

When the shortest path query structures and the shortest beer path query structures have been built for G' and G_i, $1 \leq i \leq q$, we can compute the shortest beer path or beer distance between any two vertices $u, v \in V(G)$. If $u \in V(G_i)$ and $v \in V(G_j) \setminus V(G_i)$ for some i and j, we have

$$\delta_{BG}(u, v) = \min\{M_1, M_2, M_3\}, where \tag{3}$$

$$M_1 = \min\{\delta_{BG_i}(u, x) + \delta_{G'}(x, y) + \delta_{G_j}(y, v) | x \in C(G_i), y \in C(G_j)\}, \tag{4}$$

$$M_2 = min\{\delta_{G_i}(u, x) + \delta_{BG'}(x, y) + \delta_{G_j}(y, v) | x \in C(G_i), y \in C(G_j)\}, \tag{5}$$

$$M_3 = min\{\delta_{G_i}(u, x) + \delta_{G'}(x, y) + \delta_{BG_j}(y, v) | x \in C(G_i), y \in C(G_j)\}. \tag{6}$$

If $u, v \in V(G_i)$ for some i, we have

$$\delta_{BG}(u, v) = \min\{M_1, M_2, M_3\}, where \tag{7}$$

$$M_1 = \min\{\delta_{BG_i}(u, v), \min\{\delta_{BG_i}(u, x) + \delta_{G'}(x, y) + \delta_{G_i}(y, v) | x, y \in C(G_i)\}\}, \tag{8}$$

$$M_2 = \min\{\delta_{BG_i}(u, v), \min\{\delta_{G_i}(u, x) + \delta_{BG'}(x, y) + \delta_{G_i}(y, v) | x, y \in C(G_i)\}\}, \tag{9}$$

$$M_3 = \min\{\delta_{BG_i}(u, v), \min\{\delta_{G_i}(u, x) + \delta_{G'}(x, y) + \delta_{BG_i}(y, v) | x, y \in C(G_i)\}\}. \tag{10}$$

Replacing the beer distances realizing $\delta_{BG}(u, v)$ in Equation (3) or Equation (7) by the corresponding shortest beer paths gives the shortest beer path from u to v.

An edge in $E(G)$ corresponds to an edge in exactly one graph in $\{G_i | 1 \leq i \leq q\} \cup G'$. If an edge weight is updated and the edge is in G', one only needs to update the structure for G'. If the edge is in G_i, one needs to update the structure for G_i and G'. Since an edge $\langle x, y\rangle$ with weight $\delta_{G_i}(x, y)$ and a beer edge $\langle x, y\rangle_B$ with weight $\delta_{BG_i}(x, y)$ were added in G' for each pair $x, y \in C(H_i)$, the change of an edge weight in G_i can incur at most $2(3t)^2$ edge or beer edge weight changes in G'. The procedure of graph partitioning and construction of a reduced beer graph is repeated recursively for each of $G_1, \ldots, G_q, G'$ until the component

subgraphs at the deepest recursion level have small sizes. A static shortest beer path query structure in Theorem 9 and a static shortest path query structure in Section 3 are built for each component subgraph at the deepest recursion level.

Since the dynamic data structure is built recursively, a beer distance query or a shortest beer path query is answered by recursively querying lower recursion level structures. Eventually queries are made and answered at the static query structures at the bottom level. An edge weight update is accommodated by recursively updating lower level structures. An update at a static shortest path (or shortest beer path) query structure at the bottom level is accommodated by rebuilding the static query structure.

Let $r-1$ denote the number of recursion steps. We omit the details of calculating the preprocessing/query/update times of the dynamic data structure, since the calculations are similar to the calculations for the dynamic shortest path structure in Theorem 10 and Corollary 11. The dynamic data structure is summarized in the following theorem.

▶ **Theorem 12.** *Let G be an n-vertex beer digraph with treewidth t and assume a binary tree decomposition of G with treewidth t is given. For any positive integer constant r, after $O(C(r)t^{r+3}n)$ time preprocessing, beer distance queries can be answered in $O(C(r)t^{2r+2}\alpha(n))$ time, shortest beer path queries can be answered in $O(C(r)t^{2r+2}\alpha(n) + 3^r t^3\alpha(n)L)$ time, where $C(r) = 3^{r(r+3)}$ and L is the number of edges on the shortest beer path. The data structure updates in $O(C(r)t^{2r+2}n^{(1/2)^{r-1}})$ time after an edge weight change.*[2]

Combined with Theorem 1, we obtain the following theorem by setting $r = 1 - \log\beta$.

▶ **Theorem 13.** *Let G be an n-vertex beer digraph with constant treewidth, and let β be any constant in $(0, 1)$. After $O(n)$ time preprocessing, beer distance queries can be answered in $O(\alpha(n))$ time, shortest beer path queries can be answered in $O(\alpha(n)L)$ time where L is the number of edges on the shortest beer path. Edge weight updates (including edge deletions and edge re-insertions) can be performed in $O(n^\beta)$ time.*

6 Another dynamic shortest path query structure

In this section we give another dynamic shortest path query structure. The basic structure is a balanced tree $BST(G)$ which represents a *balanced separator decomposition* of G. We build compressed graphs for the associated subgraphs of nodes in $BST(G)$ so that the distances between vertices in different separators can be quickly computed. Finally, we partition edges in $E(G)$ among nodes in $BST(G)$ so that edge weight update can be handled efficiently.

We give some preliminaries in Section 6.1, and describe the structure in Section 6.2, Section 6.3 and Section 6.4.

6.1 Preliminaries

Let (X, T) be a binary tree decomposition of G. Recall that an edge $e = (i, j)$ in $E(T)$ corresponds to $X_i \cap X_j$, which is a separator of G. We can get a decomposition of G alongside a decomposition of T. Let ϕ denote the separator that an edge e in $E(T)$ corresponds to. The removal of e from T divides T into two subtrees T_1 and T_2, the removal of ϕ from G divides G into two disjoint subgraphs $G[V_1]$ and $G[V_2]$. We add the separator ϕ and the edges in $E(G)$ between vertices in ϕ and vertices in V_1 to $G[V_1]$, and get the subgraph $G_1 = G[V_1] \cup \phi \cup E_1$

[2] One can detect whether the edge weight change incurs a negative cycle, by using Lemma 3 at the partitioned component subgraphs.

where E_1 is the set of edges in $E(G)$ joining vertices in ϕ and vertices in V_1. We add the separator ϕ and the edges in $E(G)$ between vertices in ϕ and vertices in V_2 to $G[V_2]$, and get the subgraph $G_2 = G[V_2] \cup \phi \cup E_2$ where E_2 is the set of edges in $E(G)$ joining vertices in ϕ and vertices in V_2. Thus G is divided into G_1, G_2 and $G[\phi]$ where $G[\phi]$ is the subgraph of G induced by vertices in ϕ. See Figure 2(a). We call $G[\phi]$ a *separator subgraph*. We have that T_1 is a tree decomposition of G_1 and T_2 is a tree decomposition of G_2. Note that G_1, G_2 and $G[\phi]$ partitions edges in $E(G)$.

If we repeat the division step for T_1 and G_1, and for T_2 and G_2 recursively until the subtrees have $O(1)$ nodes, we obtain a decomposition of G. We use a tree DT to represent the decomposition. Each node p of DT corresponds to a subtree of T and is associated with a subgraph of G, denoted as G_p. If p is an internal node of DT, p is also associated with an edge e_p in $E(T)$ and a separator ϕ_p which corresponds to e_p. Recall that $G_p[\phi_p]$ is called a separator subgraph, and we associate p with $G_p[\phi_p]$. Since the division stops at a leaf node of DT, a leaf node has no associated separator or separator subgraph.

Let Φ denote the set of separators associated with nodes of DT. The separators of Φ that separate G_p from the rest of G are called the *cut separators* of G_p. See Figure 2(b) for an illustration.

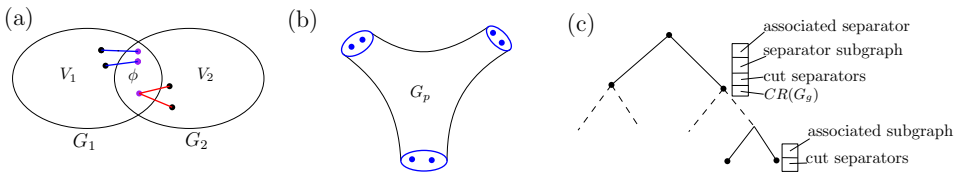


Figure 2 (a) $G_1 = G[V_1] \cup \phi \cup E_1$ where E_1 is the set of edges in $E(G)$ joining vertices in ϕ and vertices in V_1. The blue edges are in E_1. $G_2 = G[V_2] \cup \phi \cup E_2$ where E_2 is the set of edges in $E(G)$ joining vertices in ϕ and vertices in V_2. The red edges are in E_2. G_1, G_2 and $G[\phi]$ partitions edges in $E(G)$. (b) The cut separators of G_p are in blue. (c) The $BST(G)$ structure.

Constructing a balanced separator decomposition

We give an algorithm that outputs a decomposition of G that will be used in the construction of the data structure in Section 6.2. The decomposition is the construction of a DT using balanced separators. A balanced separator corresponds to an edge of a tree decomposition whose removal partitions the tree decomposition into two subtrees of proportional sizes. The decomposition fulfills two goals: (1) the height of the DT is $O(\log n)$ and (2) any subgraph associated with a node of DT has constant size separators. If in the decomposition of T we successively use an edge of a subtree whose removal partitions the subtree into two subtrees of proportional sizes, we get a *balanced decomposition* of T which fulfills the first goal.

Guibas et al. [14] showed an algorithm that computes a balanced decomposition of a binary tree in linear time. They called the balanced decomposition a *centroid decomposition*. In their algorithm the binary tree is decomposed by removing a *centroid edge* which partitions the binary tree into two subtrees, each of size at least $(|T| + 1)/3$. A centroid edge is found by finding a node that is a centroid of the tree. When each of the subtrees is partitioned similarly and this is repeated recursively until the subtrees are single nodes, we obtain a balanced binary tree structure which is the balanced decomposition of the binary tree. For our purpose, we perform a centroid decomposition on T until the subtrees have at most 5 nodes.

To fulfill the second goal, we modify the centroid decomposition procedure by using the following step when choosing the edge used to partition a subtree. Edges in $E(T)$ which are used for partitioning subtrees are called *partition edges*. If a subtree is connected to the rest of T through at most three partition edges, we just choose an centroid edge of the subtree as the partition edge (used to partition the subtree). If a subtree is connected to the rest of T through four partition edges, we choose an edge of the subtree whose removal partitions the subtree into two subtrees, either of which is connected to the rest of T through two of the four partition edges. Thus in every two levels of recursion, the size of a subtree is reduced by a factor of at least $1/3$.

If we use the above modified centroid decomposition on T to get a decomposition of G, we get a balanced DT which we call a *balanced separator tree* of G, denoted as $BST(G)$.

$BST(G)$ is a balanced decomposition of G which meets the two goals. The second goal is fulfilled since any subgraph associated with a node of $BST(G)$ has at most four cut separators, due to the fact that any subtree in the modified centroid decomposition on T is connected to the rest of T through at most four partition edges. The two goals will support the shortest path and distance queries in Section 6.3. For an internal node of $BST(G)$, we store with it its associated separator and its separator subgraph. For a leaf node of $BST(G)$, we store with its associated subgraph. For any node p of $BST(G)$, we store with it pointers to edges in $E(T)$ that correspond to the cut separators of G_p. See Figure 2(c) for an illustration.

Assume that a binary tree decompostion (X, T) of G with treewidth t is given. The modified centroid decomposition on T takes $O(n)$ time. Storing the associated separators and separator subgraphs at internal nodes of $BST(G)$ takes $O(t^2 n)$ time. The associated subgraph of a leaf node in $BST(G)$ has $O(t)$ vertices, so storing the associated subgraphs at leaf nodes of $BST(G)$ takes $O(t^2 n)$ time. In summary:

▶ **Lemma 14.** *Given a binary tree decomposition (X, T) of G with treewidth t, we can build $BST(G)$ in $O(t^2 n)$ time. The depth of $BST(G)$ is $O(\log n)$. The associated subgraph of a node in $BST(G)$ has at most four cut separators.*

6.2 The preprocessing

Given a binary tree decompostion (X, T) of G, we preprocess T and G to output the following data structures:

1. The $BST(G)$ with $CR(G_g)$ for each internal node g in $BST(G)$. $CR(G_g)$ is a compressed representation of g's associated subgraph G_g, which will be described below.
2. An array $Loc[\cdot]$ for vertices in $V(G)$, where $Loc[v]$ is a leaf node of $BST(G)$ whose associated subgraph contains vertex v in $V(G)$.
3. An LCA (lowest common ancestor) structure for $BST(G)$.

We have the following definition.

▶ **Definition 15.** *Let $BST(G)$ be the balanced separator tree of G and let p be a node of $BST(G)$. If p is an internal node g, let $CR(G_g)$ be a clique for G_g with respect to vertices in ϕ_g and the cut separators of G_g. The weight of an edge in the clique is the distance in G_g between the vertices of the edge. We call $CR(G_g)$ the compressed representation of G_g. If p is a leaf node, we have $CR(G_p) = G_p$.*

Given $BST(G)$, we compute $CR(G_g)$ for each internal node g of $BST(G)$ in a bottom-up fashion. Let t_1 and t_2 be g's children. We take the union of $CR(G_{t_1})$, $CR(G_{t_2})$ and g's associated separator subgraph $G_g[\phi_g]$, run the Floyd-Warshall algorithm on the union graph, and build a clique on vertices in ϕ_g and the cut separators of G_g where the weight of an edge is the distance computed for the vertices of the edge.

The correctness of the clique construction procedure is supported by the following lemma.

▶ **Lemma 16.** *lemmaCRunion Let g be an internal node of $BST(G)$ and let ϕ_g be its associated separator. Let t_1 and t_2 be g's children in $BST(G)$. Let G_{t_1} be t_1's associated subgraph and let G_{t_2} be t_2's associated subgraph. The distance between any two vertices in $CR(G_{t_1}) \cup CR(G_{t_2}) \cup G_g[\phi_g]$ equals the distance between the two vertices in G_g.*

Note that the associated subgraphs of leaf nodes in $BST(G)$ and the separator subgraphs of internal nodes in $BST(G)$ *partition* edges in $E(G)$. This property is used for efficiently updating an edge weight in Section 6.4.

The preprocessing consists of two steps: building the $BST(G)$, and constructing the $CR(G_g)$ for each internal node g of $BST(G)$. The associated subgraph of an internal node g in $BST(G)$ has at most four cut separators, each of which contains at most t vertices. Thus $CR(G_g)$ has $O(t)$ vertices and computing $CR(G_g)$ takes $O(t^3)$ time, dominated by the Floyd-Warshall algorithm. There are $O(n)$ internal nodes, so the second step takes $O(t^3 n)$ time. Combined with Lemma 14, we have the following lemma.

▶ **Lemma 17.** *Given a binary tree decomposition (X, T) of G with treewidth t, we can build $BST(G)$ and $CR(G_g)$ for each internal node g of $BST(G)$ in $O(t^3 n)$ time.*

The array $Loc[\cdot]$ for vertices in $V(G)$ can be constructed by scanning the associated subgraphs of leaf nodes of $BST(G)$. There are $O(n)$ leaf nodes. The associated subgraph of a leaf node has $O(t)$ vertices. Thus $Loc[\cdot]$ can be constructed in $O(tn)$ time.

We can build an LCA structure for $BST(G)$ in $O(n)$ time [3, 16].

▶ **Corollary 18.** *The preprocessing outputs the $BST(G)$ with $CR(G_g)$, $Loc[\cdot]$ and an LCA structure for $BST(G)$ in $O(t^3 n)$ time.*

6.3 Answering shortest path and distance queries

We describe the distance query algorithm first. Let u and v be any two vertices of G. The distance query algorithm consists of three steps: (1) find the node s of $BST(G)$ whose associated separator separates u from v, (2) compute the distances in G between any two vertices in ϕ_a, where ϕ_a is the associated separator of an ancestor node a of s, and (3) compute the distances in G from u to vertices in ϕ_s and the distances in G from vertices in ϕ_s to v, from which the distance from u to v is computed.

In step 1 we use the array $Loc[\cdot]$ to locate the leaf node $l(u)$ of $BST(G)$ whose associated subgraph contains u and locate the leaf node $l(v)$ whose associated subgraph contains v. We use the LCA structure on $BST(G)$ to find the vertex s, which is the lowest common ancestor of $l(u)$ and $l(v)$ in $BST(G)$.

Step 2 makes use of the compressed representations. Let g be an internal node of $BST(G)$ and let $D(\phi_g)$ denote the distances in G between any two vertices in ϕ_g. Let r be the root of $BST(G)$ and let a be an ancestor of s. We compute the distances top-down, for nodes on the path from r to s. $CR(G)$ at r is a clique containing the distances in G between any two vertices in ϕ_r, thus $D(\phi_r)$ has already been computed. Next let b be a's child that is on the path from r to s and assume we have computed $D(\phi_r), \ldots, D(\phi_a)$. We can use $CR(G_b)$ and $D(\phi_r), \ldots, D(\phi_a)$ to compute $D(\phi_b)$. The associated subgraph G_b of b is separated from the rest of G by at most four cut separators, which are among $\{\phi_r, \ldots, \phi_a\}$. Recall that we have stored, with each node p of $BST(G)$, pointers to the cut separators of G_p. If we add to $CR(G_b)$ edges with weights equal to distances in $D(\phi_x)$, where ϕ_x is a cut separator of G_b,

the distance between any two vertices in the resulting graph equals the distance in G between the two vertices. This is because absorption preserves distances in G (see Definition 2). Thus we can run the Floyd-Warshall algorithm on the resulting graph to compute $D(\phi_b)$ in $O(t^3)$ time, recalling that any compressed representation has $O(t)$ vertices. Therefore we can compute $D(\phi_r), \ldots, D(\phi_s)$ top-down from r to s, where each $D(\cdot)$ is computed in $O(t^3)$ time.

In step 3 we compute the distances in G from u to vertices in ϕ_s recursively. Computing the distances in G from vertices in ϕ_s to v is symmetric so we only discuss the former. Let s_1 be the child of s that is on the path from s to $l(u)$. We have computed $D(\phi_r), \ldots, D(\phi_q)$ in step 2, where q is the parent of s. Let $D(u, \phi_s)$ denote the distances in G from u to vertices in ϕ_s. We compute $D(u, \phi_s)$ at s_1 and differentiate between two cases:

Case 1: ϕ_{s_1} separates u from ϕ_s in G_{s_1}. See Figure 3(a). For a vertex v_s in ϕ_s, we have

$$d_G(u, v_s) = \min_{v_{s_1} \in \phi_{s_1}} \{d_G(u, v_{s_1}) + d_G(v_{s_1}, v_s)\}. \tag{11}$$

Let $D(u, \phi_{s_1}) = \{d_G(u, v_{s_1}) | v_{s_1} \in \phi_{s_1}\}$ and let $D(\phi_{s_1}, \phi_s) = \{d_G(v_{s_1}, v_s) | v_{s_1} \in \phi_{s_1}, v_s \in \phi_s\}$. Given $D(\phi_r), \ldots, D(\phi_s)$, we add edges to $CR(G_{s_1})$ with weights equal to distances in $D(\phi_x)$, where ϕ_x is a cut separator of G_{s_1}. Then we run the Floyd-Warshall algorithm on the resulting graph to compute $D(\phi_{s_1})$ and $D(\phi_{s_1}, \phi_s)$. Thus in this case, computing $D(u, \phi_s)$ is reduced to computing $D(u, \phi_{s_1})$, which is computed recursively at s' where s' is the child of s_1 on the path from s to $l(u)$.

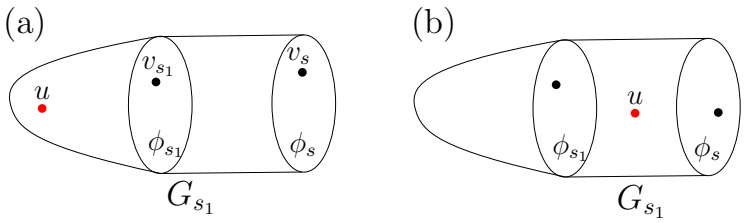


Figure 3 (a) ϕ_{s_1} separates u from ϕ_s in G_{s_1}. (b) ϕ_{s_1} does not separate u from ϕ_s in G_{s_1}.

Case 2: ϕ_{s_1} does not separate u from ϕ_s in G_{s_1}. See Figure 3(b). For this case, we add edges to $CR(G_{s_1})$ with weights equal to distances in $D(\phi_x)$, where ϕ_x is a cut separator of G_{s_1}. Then we run the Floyd-Warshall algorithm on the resulting graph to compute $D(\phi_{s_1})$. We recursively compute $D(u, \phi_s)$ at s', since ϕ_s is a cut separator of $G_{s'}$.

Recursion stops when we arrive at $l(u)$. Since we have computed $D(\phi_r), \ldots, D(\phi_y)$ where y is the parent of $l(u)$, we can add edges and run the Floyd-Warshall algorithm to compute $D(u, \phi_y)$ or $D(u, \phi_s)$, as desired.

Step 1 takes $O(1)$ time. Step 2 computes $D(\phi_r), \ldots, D(\phi_s)$ top-down from r to s, where each $D(\cdot)$ is computed in $O(t^3)$ time. Since $BST(G)$ has depth $O(\log n)$, step 2 takes $O(t^3 \log n)$ time. Each recursion step in step 3 takes $O(t^3)$ time, dominated by running the Floyd-Warshall algorithm. The computation at $l(u)$ also takes $O(t^3)$ time. Thus step 3 takes $O(t^3 \log n)$ time. In summary:

▶ **Lemma 19.** *Let u and v be any two vertices in G. The distance in G from u to v can be computed in $O(t^3 \log n)$ time where t is the treewidth of G.*

6.3.1 Shortest path query algorithm

To answer shortest path queries efficiently, we add a preprocessing step *PathExtract_Pre*. *PathExtract_Pre* works on $BST(G)$ bottom-up and stores with each internal node g the complete APSP information when running the Floyd-Warshall algorithm. In analogy to the

distance query algorithm, the shortest path query algorithm works down the path from the root to $l(u)$ and retains APSP information when running the Floyd-Warshall algorithm. The APSP information allows us to extract the shortest path in G between any two vertices in $CR(G_a)$ where a is an ancestor of $l(u)$, in time linear to the number of edges on the shortest path. We have the following lemma.

▶ **Lemma 20.** *Let u and v be any two vertices in G. One can preprocess G in $O(t^3 n)$ time and with $O(t^3 n)$ space so that the shortest path in G from u to v can be reported in $O(t^3 \log n + L)$ time where L is the number of edges on the shortest path.*

6.4 Handling edge weight update

We have the following lemma.

▶ **Lemma 21.** *The query structure can be updated in $O(t^3 \log n)$ time for an edge weight update.*

Combined with Corollary 18, Lemma 19 and Lemma 20, we obtain the main theorem of this section.

▶ **Theorem 22.** *Let G be an n-vertex digraph of treewidth t and assume a binary tree decomposition of G with treewidth t is given. One can preprocess G in $O(t^3 n)$ time and with $O(t^3 n)$ space, so that distance queries can be answered in $O(t^3 \log n)$ time, and shortest path queries can be answered in time $O(t^3 \log n + L)$, where L is the number of edges on the path. The data structure can be updated in $O(t^3 \log n)$ time for an edge weight update.*

7 A dynamic shortest beer path query structure

We can extend the structure in Section 6 to handle shortest beer paths. The general idea is to add beer edges and to compute not only APSP, but also APSBP (all pairs shortest beer path). The query structure is summarized in the following theorem.

▶ **Theorem 23.** *Let G be an n-vertex beer digraph with treewidth t and assume a binary tree decomposition of G with treewidth t is given. One can preprocess G in $O(t^3 n)$ time and with $O(t^3 n)$ space, so that beer distance queries can be answered in $O(t^3 \log n)$ time, shortest beer path queries can be answered in $O(t^3 \log n + L)$ time where L is the number of edges on the shortest beer path. The query structure can be updated in $O(t^3 \log n)$ time for an edge weight update.*

References

1 Noga Alon and Baruch Schieber. Optimal preprocessing for answering on-line product queries. Technical Report No.71/87, Tel Aviv University, 1987.

2 Joyce Bacic, Saeed Mehrabi, and Michiel Smid. Shortest beer path queries in outerplanar graphs. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC)*, volume 212 of *LIPIcs*, pages 62:1–62:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

3 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 88–94, 2000.

4 Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 226–234, 1993.

5 Panagiotis Charalampopoulos, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Almost optimal distance oracles for planar graphs. In *Proceedings of the 51st Symposium on Theory of Computing (STOC)*, pages 138–151. ACM, 2019.

6 Shiva Chaudhuri and Christos D. Zaroliagis. Shortest paths in digraphs of small treewidth. part I: sequential algorithms. *Algorithmica*, 27(3):212–226, 2000.

7 Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2:337–361, 1987.

8 Danny Z. Chen and Jinhui Xu. Shortest path queries in planar graphs. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 469–478. ACM, 2000.

9 Vincent Cohen-Addad, Søren Dahlgaard, and Christian Wulff-Nilsen. Fast and compact exact distance oracle for planar graphs. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 962–973. IEEE Computer Society, 2017.

10 Hristo N. Djidjev. On-line algorithms for shortest path problems on planar digraphs. In *Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 1197 of *Lecture Notes in Computer Science*, pages 151–165, 1996.

11 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, near linear time. In *42nd Annual Symposium on Foundations of Computer Science, (FOCS)*, pages 232–241, 2001.

12 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3):34:1–34:45, 2018.

13 Viktor Fredslund-Hansen, Shay Mozes, and Christian Wulff-Nilsen. Truly subquadratic exact distance oracles with constant query time for planar graphs. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC)*, volume 212 of *LIPIcs*, pages 25:1–25:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

14 Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert Endre Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

15 Tesshu Hanaka, Hirotaka Ono, Kunihiko Sadakane, and Kosuke Sugiyama. Shortest beer path queries based on graph decomposition, 2023. `arXiv:2307.02787`.

16 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

17 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–192, 2021.

18 Yaowei Long and Seth Pettie. Planar distance oracles with better time-space tradeoffs. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2517–2537. SIAM, 2021.

19 Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–222. SIAM, 2012.

20 Seth Pettie. An inverse-ackermann type lower bound for online minimum spanning tree verification. *Combinatorica*, 26(2):207–230, 2006.

21 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. In *Proceedings of the 12th Annual European Symposium (ESA)*, volume 3221 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 2004.

22 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

23 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 645–654. IEEE Computer Society, 2010.

Coloring and Recognizing Mixed Interval Graphs

Grzegorz Gutowski
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

Konstanty Junosza-Szaniawski
Warsaw University of Technology, Poland

Felix Klesen
Universität Würzburg, Germany

Paweł Rzążewski
Warsaw University of Technology, Poland
Institute of Informatics, University of Warsaw, Poland

Alexander Wolff
Universität Würzburg, Germany

Johannes Zink
Universität Würzburg, Germany

Abstract

A *mixed interval graph* is an interval graph that has, for every pair of intersecting intervals, either an arc (directed arbitrarily) or an (undirected) edge. We are particularly interested in scenarios where edges and arcs are defined by the geometry of intervals. In a proper coloring of a mixed interval graph G, an interval u receives a lower (different) color than an interval v if G contains arc (u, v) (edge $\{u, v\}$). Coloring of mixed graphs has applications, for example, in scheduling with precedence constraints; see a survey by Sotskov [Mathematics, 2020].

For coloring general mixed interval graphs, we present a $\min\{\omega(G), \lambda(G) + 1\}$-approximation algorithm, where $\omega(G)$ is the size of a largest clique and $\lambda(G)$ is the length of a longest directed path in G. For the subclass of *bidirectional interval graphs* (introduced recently for an application in graph drawing), we show that optimal coloring is NP-hard. This was known for general mixed interval graphs.

We introduce a new natural class of mixed interval graphs, which we call *containment interval graphs*. In such a graph, there is an arc (u, v) if interval u contains interval v, and there is an edge $\{u, v\}$ if u and v overlap. We show that these graphs can be recognized in polynomial time, that coloring them with the minimum number of colors is NP-hard, and that there is a 2-approximation algorithm for coloring.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Interval Graphs, Mixed Graphs, Graph Coloring

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.36

Related Version *Full Version*: https://arxiv.org/abs/2303.07960

Funding *Grzegorz Gutowski*: partially supported by the National Science Center of Poland under grant no. 2019/35/B/ST6/02472.
Johannes Zink: partially supported by DFG grant Wo 758/11-1.

Acknowledgements We are indebted to Krzysztof Fleszar, Zbigniew Lonc, Karolina Okrasa, and Marta Piecyk for fruitful discussions. Additionally, we acknowledge the welcoming and productive atmosphere at the workshop Homonolo 2022, where some of the work was done.

1 Introduction

In a geometric intersection graph, the vertices represent geometric objects, and two vertices are adjacent if and only if the corresponding objects intersect. For example, *interval graphs* are the intersection graphs of intervals on the real line. These graphs are well understood: interval graphs are *chordal* and can thus be colored optimally (that is, with the least number of colors) in polynomial time. In other words, given an interval graph G, its *chromatic number* $\chi(G)$ can be computed efficiently.

The notion of coloring can be adapted to directed graphs where an arc (u, v) means that the color of u must be smaller than that of v. Clearly, such a coloring can only exist if the given graph is acyclic. Given a directed acyclic graph, its chromatic number can be computed efficiently (via topological sorting).

A generalization of both undirected and directed graphs are *mixed graphs* that have edges and arcs. A *proper coloring* of a mixed graph G with vertex set $V(G)$ is a function $f\colon V(G) \to \mathbb{N}$ such that, for any distinct vertices u and v of G, the following conditions hold:

1. if there is an edge $\{u, v\}$, then $f(u) \neq f(v)$, and
2. if there is an arc (u, v), then $f(u) < f(v)$.

The objective is to minimize the number of colors.

The concept of mixed graphs was introduced by Sotskov and Tanaev [16] and reintroduced by Hansen, Kuplinsky, and de Werra [9] in the context of proper colorings of mixed graphs. Properly coloring mixed graphs is NP-hard even for bipartite planar graphs [12] but admits efficient algorithms for trees [5] and series-parallel graphs [6].

For a mixed interval graph G, the *underlying undirected graph* of G, denoted by $U(G)$, has an edge for every edge or arc of G. Note that testing whether a given graph G is a mixed interval graph means testing whether $U(G)$ is an interval graph, which takes linear time [10].

Motivation. Mixed graphs are graphs where some vertices are connected by (undirected) edges and others by directed arcs. Such structures are useful for modeling relationships that involve both directed and undirected connections and find applications in various areas, including network analysis, transportation planing, job scheduling, and circuit design.

Coloring mixed graphs is relevant in task scheduling problems where tasks have dependencies and resource requirements [18, 3, 15]. In circuit design, coloring mixed graphs allows us to reduce signal crosstalk or interference. Other applications include modeling of metabolic pathways in biology [1], process management in operating systems [2], traffic signal synchronization [13], and timetabling [4]. See the extensive survey by Sotskov [14] for other applications and relevant problems. Coloring of mixed graphs is a challenging problem, as many techniques known for solving graph coloring problems fail in the more general setting.

The study of mixed graph coloring for interval graphs was initiated by Zink et al. [19], motivated by the minimization of the number of additional sub-layers for routing edges in layered orthogonal graph drawing according to the so-called Sugiyama framework [17]. In a follow-up paper, Gutowski et al. [8] resolved some of the problems concerning interval graphs where the subset of arcs and their orientations are given by the geometry of the intersecting intervals. Driven by the graph drawing application, they focused on the *directional variant* where, for every pair of intersecting intervals, there is an edge when one interval is contained in the other and there is an arc oriented towards the right interval when the intervals overlap.

In this paper we focus on the *containment variant* where, for any pair of intersecting intervals, there is an arc oriented towards the smaller interval when one interval is contained in the other, and an edge when they overlap. This is the only other natural geometric variant

Table 1 Known and new results concerning subclasses of mixed interval graphs. The time complexities refer to a given set of n intervals with m pairwise intersections. (We use T, P, and C as shorthand for Theorem, Proposition, Corollary, respectively.)

Mixed interval graph class	Coloring								Recognition	
	complexity		lower bound		upper bound		approximation			
containment	NP-hard	T7	$2\omega-1$	P6	$2\omega-1$	T2	2	C5	$O(nm)$	T1
directional	$O(n \log n)$	[8]					1	[8]	$O(n^2)$	[8]
bidirectional	NP-hard	T8					2	[8]	open	
general	NP-hard	[8]	$(\lambda+2)\omega/2$	P10	$(\lambda+1)\omega$	T9	$\min\{\omega, \lambda+1\}$	T9	$O(n+m)$	[10]

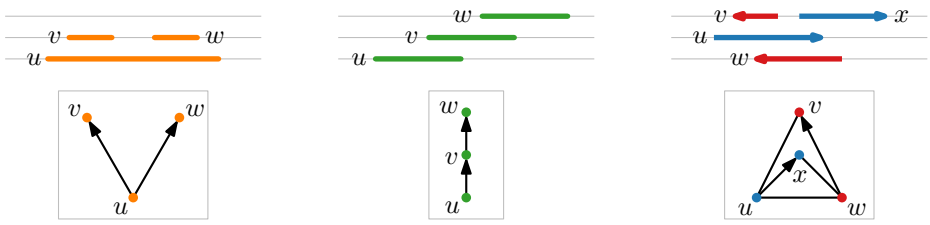
that can be defined for interval graphs, but the containment variant can also be defined for other geometric intersection graphs, or even for graphs defined by systems of intersecting sets.

As there are already some effective techniques for the directional variant, our hope was to use them in the containment variant, or even in more general settings. Quite unexpectedly, the containment variant for interval graphs turned out to be more difficult than the directional variant. We have found our techniques for proving lower bounds for the containment variant of interval graphs to be applicable for the *bidirectional variant* considered previously [19, 8]. This variant is a generalization of the directional variant mentioned above. Every interval has an orientation; left-going or right-going. There is an arc between two intervals if and only if they overlap and their orientations agree. Arcs between left-going intervals are directed as in the directional variant; the condition for right-going intervals is symmetric. As a result, we get that minimizing the number of additional sub-layers in layered orthogonal graph drawing according to the Sugiyama framework is NP-hard.

Our Contribution. In this paper we forward the study of coloring mixed graphs where edge directions have a geometric meaning. To this end, we introduce a new natural class of mixed interval graphs, which we call *containment interval graphs*. In such a graph, there is an arc (u, v) if interval u contains interval v, and there is an edge $\{u, v\}$ if u and v overlap. For a set $\mathcal{I}$ of intervals, let $\mathcal{C}[\mathcal{I}]$ be the containment interval graph induced by $\mathcal{I}$. We show that these graphs can be recognized in polynomial time (Section 2), that coloring them optimally is NP-hard (Section 4), and that, for every set $\mathcal{I}$ of intervals, it holds that $\chi(\mathcal{C}[\mathcal{I}]) \leq 2\omega(\mathcal{C}[\mathcal{I}]) - 1$, that is, $\mathcal{C}[\mathcal{I}]$ can be colored with fewer than twice as many colors as the size of the largest clique in $\mathcal{C}[\mathcal{I}]$ (Section 3). In other words, containment interval graphs are χ-*bounded*. Our constructive proof yields a 2-approximation algorithm for coloring containment interval graphs.

Then we prove that, for the class of bidirectional interval graphs, optimal coloring is NP-hard (Section 5). This answers (negatively) an open problem that was asked previously [8]. Our reduction is similar to the one for containment interval graphs, but technically somewhat more challenging. Finally, we show that, for any mixed interval graph G without directed cycles, it holds that $\chi(G) \leq \omega(G) \cdot (\lambda(G) + 1)$, where $\lambda(G)$ denotes the length of a longest directed path in G (Section 6). Since $\chi(G) \geq \max\{\omega(G), \lambda(G) + 1\}$, our constructive proof for the upper bound yields a $\min\{\omega(G), \lambda(G) + 1\}$-approximation algorithm. The upper bound is asymptotically tight in the worst case.

Table 1 gives an overview over known and new results concerning the above-mentioned subclasses of mixed interval graphs. Given a positive integer k, we use $[k]$ as shorthand for the set $\{1, 2, \ldots, k\}$. When we visualize a graph coloring corresponding to a set of intervals,



(a) $G_1 \in \mathcal{C} \setminus \mathcal{B}$. **(b)** $G_2 \in \mathcal{D} \setminus \mathcal{C}$. **(c)** $G_3 \in \mathcal{B} \setminus \mathcal{D}$.

Figure 1 Let $\mathcal{D}$, $\mathcal{B}$, and $\mathcal{C}$ be the classes of directional, bidirectional, and containment interval graphs. Clearly, $\mathcal{D} \subseteq \mathcal{B}$. The above sets of intervals and the corresponding directed graphs show that the classes $\mathcal{D}$ and $\mathcal{B}$ are incomparable with the class $\mathcal{C}$ and that $\mathcal{D}$ is properly contained in $\mathcal{B}$.

we use horizontal tracks to indicate the color. In Figure 1, we briefly analyze the relationships between the three classes of geometrically defined mixed interval graphs; directional ($\mathcal{D}$), bidirectional ($\mathcal{B}$), and containment interval graphs ($\mathcal{C}$).

2 Recognition of Containment Interval Graphs

Booth and Lueker [10] introduced a data structure called *PQ-tree* to recognize, for a given undirected graph G, whether G is an interval graph. A PQ-tree is a rooted tree of so-called *P-nodes*, where the order of children can be arbitrarily permuted, and *Q-nodes*, where the order of children is fixed up to inversion. A specific permutation of all nodes is called a *rotation*. One can think of the leaves of a PQ-tree to represent the maximal cliques of G and a specific rotation to represent an order of the maximal cliques, which implies an interval representation of G where every vertex is contained in a consecutive sequence of maximal cliques. (Actually, a PQ-tree can encode all possible interval representations of G.)

Observe that a representation of a containment interval graph is an interval representation. Hence, if, for a given mixed graph G, a containment representation $\mathcal{I}$ exists, then $\mathcal{I}$ corresponds to a rotation of the PQ-tree constructed for the underlying undirected graph $U(G)$ by the algorithm of Booth and Lueker [10]. Hence, to recognize a containment interval graph G, we proceed in three phases. First, we compute a PQ-tree T of $U(G)$. Second, we find a rotation of T corresponding to a containment representation of G. Third, we determine suitable endpoints for the intervals corresponding to our selected rotation resulting in a containment representation $\mathcal{I}$ of G.

In the second phase, we proceed top-down to fix the permutation of each node of T while we maintain as invariant that before and after deciding the permutation of a single node, we can still reach a rotation of T corresponding to a containment representation $\mathcal{I}$ (provided G is a containment interval graph). Depending on the set of maximal cliques (corresponding to leaves) a vertex v is contained in, we can determine where v is *introduced* in T (roughly at the root of the subtree containing all leaves corresponding to v). Intuitively, it is "natural" for a vertex u introduced further up in the tree to have an arc towards a vertex v introduced further down in the tree. However, if u and v are connected by an edge, we need to permute the nodes of the PQ-tree such that both u and v start or end in the same maximal clique. These restrictions can propagate.

If we end up with a rotation of T, we construct, in the third phase, a corresponding containment representation if possible. To this end, we determine for every vertex the first and the last clique it appears in, which groups the left and right endpoint of the intervals. Within each group, we sort the endpoints according to the constraints implied by the arcs

and edges where possible. What remains are induced mixed subgraphs of vertices that start and end in the same cliques and that behave the same with respect to every other vertex (i.e., they are all connected to this vertex by an outgoing arc or an incoming arc or an edge). We can interpret each such subgraph as a *partially ordered set* for which we need to check whether it is *two-dimensional* and find two corresponding linear orders, which gives us an ordering of their left and their right endpoints. This last part depends on the linear-time algorithm by McConnell and Spinrad [11] that can construct such two orders for any two-dimensional poset.

▶ **Theorem 1.** *There is an algorithm that, given a mixed graph G, decides whether G is a containment interval graph. The algorithm runs in $O(nm)$ time, where n is the number of vertices of G and m is the total number of edges and arcs of G, and produces a containment representation of G if G admits one.*

The full proof follows the ideas presented above, but has some technical subtleties; see the full version of this article [7].

3 A 2-Approximation Algorithm for Coloring Containment Interval Graphs

In this section, we present a 2-approximation algorithm for coloring containment interval graphs, we detail how to make the algorithm run in $O(n \log n)$ time for a set of n intervals, and we construct a family of sets of intervals that shows that our analysis is tight.

▶ **Theorem 2.** *For any set $\mathcal{I}$ of intervals, the containment interval graph $\mathcal{C}[\mathcal{I}]$ induced by $\mathcal{I}$ admits a proper coloring with at most $2 \cdot \omega(\mathcal{C}[\mathcal{I}]) - 1$ colors.*

Proof. For simplicity, let $G := \mathcal{C}[\mathcal{I}]$ and $\omega := \omega(G)$. We use induction on ω. If $\omega = 1$, then G has no edges and clearly admits a proper coloring using only one color. So assume that $\omega > 1$ and that the theorem holds for all graphs with smaller clique number.

Recall that a *proper* interval graph is an interval graph that has a representation where no interval is contained in another interval. Let $M(\mathcal{I})$ denote the subset of $\mathcal{I}$ consisting of intervals that are maximal with respect to the containment relation. In particular, $\mathcal{C}[M(\mathcal{I})]$ is a proper interval graph. Observe that $\bigcup M(\mathcal{I}) = \bigcup \mathcal{I}$ (where we consider the union of intervals as a subset of the real line). Let R be an inclusion-wise minimal subset of $M(\mathcal{I})$ such that $\bigcup R = \bigcup \mathcal{I}$. In Figure 2, the intervals in $M(\mathcal{I})$ are marked with crosses and the set of intervals on the lowest two (gray) lines is one way of choosing R.

▷ Claim 3. $\mathcal{C}[R]$ is an undirected linear forest.

Proof. All intervals in $M(\mathcal{I})$ and thus in R are incomparable with respect to the containment relation, so $\mathcal{C}[R]$ has no arcs. Note that $\mathcal{C}[R]$ is a proper interval graph, so it contains no induced $K_{1,3}$ and no induced cycle with at least four vertices. Thus it suffices to prove that $\mathcal{C}[R]$ is triangle-free. For contradiction, suppose otherwise. Let x, y, z induce a triangle in $\mathcal{C}[R]$, ordered according to their left endpoints. As x, y, z are pairwise overlapping, note that $y \subseteq x \cup z$, and thus $\bigcup(R \setminus \{y\}) = \bigcup R$. This contradicts the minimality of R. ◁

By the claim above, $\mathcal{C}[R]$ can be properly colored with colors $\{1, 2\}$. Let f_1 be such a coloring. If $R = \mathcal{I}$, we are done (using only ω many colors), so suppose that $\mathcal{I} \setminus R \neq \emptyset$. Slightly abusing notation, we define $G' := G - R$.

▷ Claim 4. The largest clique in G' has at most $\omega - 1$ vertices.

Figure 2 A set of intervals and a coloring produced by the 2-approximation algorithm. The intervals that lie in $M(\mathcal{I})$ at the top level of the recursion are marked with crosses.

Proof. As G' is a subgraph of G, each clique in G' has at most ω vertices. For contradiction, suppose that there is a set $S \subseteq \mathcal{I} \setminus R$ such that $|S| = \omega$ and all intervals in S pairwise intersect. By the Helly property of intervals, $\bigcap S \neq \emptyset$. Let $p \in \bigcap S$. Since $\bigcup R = \bigcup \mathcal{I}$, there is an interval $r \in R$ that contains p. Thus $S \cup \{r\}$ is a clique in G with $\omega + 1$ vertices, which contradicts the definition of ω. ◁

By the inductive assumption, G' admits a proper coloring f_2 using colors $[2(\omega - 1) - 1]$. Finally, we define $f : \mathcal{I} \to [2\omega - 1]$ as follows:

$$f(x) = \begin{cases} f_1(x) & \text{if } x \in R, \\ f_2(x) + 2 & \text{if } x \notin R. \end{cases}$$

We claim that f is a proper coloring of G. (For an example, see Figure 2.)

First, note that if $x, y \in \mathcal{I}$ are distinct and $x \cap y \neq \emptyset$, then $f(x) \neq f(y)$. Indeed, if $x, y \in R$, then $f(x) = f_1(x) \neq f_1(y) = f(y)$. If $x, y \notin R$, then $f(x) = f_2(x) + 2 \neq f_2(y) + 2 = f(y)$. Finally, if, say, $x \in R$ and $y \notin R$, then $f(x) \in \{1, 2\}$ and $f(y) \geq 3$.

It remains to argue that the second condition in the definition of a proper coloring holds as well. For a contradiction, let x and y be distinct intervals and assume that $x \subseteq y$ and $f(y) > f(x)$. Note that $x \notin M(\mathcal{I})$ and thus $x \notin R$. This implies that $f(x) \geq 3$. Since we assumed that $f(y) > f(x)$, we have that $f(y) > 3$. Hence, $y \notin R$. However, by the inductive assumption, we have that $f(x) = f_2(x) + 2 > f_2(y) + 2 = f(y)$, which yields the desired contradiction. This completes the proof. ◀

Observe that the proof of Theorem 2 can be easily transformed into an efficient algorithm, which yields the following corollary.

▶ **Corollary 5.** *There is a 2-approximation algorithm for coloring interval containment graphs properly. Given a set of n intervals, the algorithm runs in $O(n \log n)$ time.*

Proof. For any graph G, we have $\chi(G) \geq \omega(G)$. Hence, the approximation factor follows directly from Theorem 2.

It remains to implement the constructive proof of Theorem 2 efficiently. Let $\mathcal{I}$ be the given set of intervals. For each interval I in $\mathcal{I}$, let r_I be the right endpoint of I. We go through the intervals from left to right in several phases. In each phase, we use two colors, except possibly in the last phase where we may use only one color. For phase i with $i \geq 1$, we reserve the set $\mathit{colors}(i) = \{2i - 1, 2i\}$. We use an augmented balanced binary search tree $\mathcal{T}$ to store the intervals in $\mathcal{I}$. We will query $\mathcal{T}$ in two ways. A query of type Q1 in $\mathcal{T}$ with a value $x \in \mathbb{R} \cup \{-\infty\}$ will return, among all intervals whose left endpoint is at least x, one with leftmost left endpoint (and *nil* if such an interval does not exist). A query of type Q2 in $\mathcal{T}$ with a value $y \in \mathbb{R}$ will return, among all intervals whose left endpoint is at most y, one with rightmost right endpoint (and *nil* if such an interval does not exist). Note that the two queries are not symmetric.

Algorithm 1 describes our algorithm in pseudocode. Initially, $\mathcal{T}$ stores all intervals in $\mathcal{I}$. The algorithm terminates once $\mathcal{T}$ is empty and all intervals are colored.

Algorithm 1 2-Approximate-Coloring(set $\mathcal{I}$ of n intervals).

```
T.initialize(I)
i = 0
while not T.empty() do
    i = i + 1                                              // start new phase
    I = T.Q1(−∞)
    c = I.color = 2i − 1                          // color I and set current color
    T.remove(I)
    while not T.empty() do
        I′ = T.Q2(r_I)
        if I′ == nil or r_{I′} ≤ r_I then                  // no interval contains r_I
            I′ = T.Q1(r_I)
            if I′ == nil then break                        // finish current phase
            I′.color = c                          // color I′ and keep current color
        else                                                // I and I′ overlap
            c = I′.color = colors(i) \ {c}        // color I′ and swap current color
        T.remove(I′)
        I = I′
```

We start each phase by Q1-querying $\mathcal{T}$ with $-\infty$. This yields the leftmost interval I stored in $\mathcal{T}$. We color I with the smaller color $2i-1$ reserved for the current phase i. Let the *current color* c be this color. We remove I from $\mathcal{T}$. Then we Q2-query $\mathcal{T}$ with the right endpoint r_I of I and consider the following two possibilities.

Case I: If the Q2-query returns *nil* or an interval that lies completely to the left of r_I, we Q1-query $\mathcal{T}$ with r_I for an interval to the right of r_I. If such an interval I' exists, it must be disjoint from I, so we color I' with the current color c. Otherwise, we start a new phase.

Case II: If the Q2-query returns an interval I' that overlaps with the previous interval I, we color I' with the other color c' that we reserved for the current phase, that is, $\{c'\} = \mathit{colors}(i) \setminus \{c\}$. Then we set the current color c to c'.

In either case, if we do not start a new phase, we remove I' from $\mathcal{T}$ and proceed with the next Q2-query as above, with I' now playing the role of I.

It remains to implement the balanced binary search tree $\mathcal{T}$. The key of an interval is its left endpoint. For simplicity, we assume that the intervals are stored in the leaves of $\mathcal{T}$ and that the key of each inner node is the maximum of the keys in its left subtree. This suffices to answer queries of type Q1. For queries of type Q2, we augment $\mathcal{T}$ by storing, with each node ν, a value $\max(\nu)$ that we set to the maximum of the right endpoints among all intervals in the subtree rooted at ν. (We also store a pointer $\mu(\nu)$ to the interval that yields the maximum.) In a Q2-query with a value y, we search for the largest key $k \le y$. Let π be the search path in $\mathcal{T}$, and initialize m with $-\infty$. When traversing π, we inspect each node ν that hangs off π on the left side. If $\max(\nu) > m$, then we set $m = \max(\nu)$ and $\rho = \mu(\nu)$. When we reach a leaf, ρ points to an interval whose right endpoint is maximum among all intervals whose left endpoint is at most y.

The runtime of $O(n \log n)$ is obvious since we insert, query, and delete each interval in $O(\log n)$ time exactly once. ◀

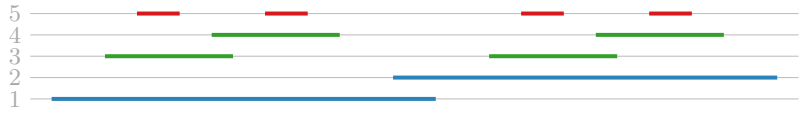


Figure 3 Instance for the proof of Proposition 6 for $n = 3$: $|\mathcal{I}_n| = 3 \cdot 2^{n-1} - 2 = 10$, $\omega(\mathcal{I}_n) = n = 3$, and $\chi(\mathcal{I}_n) = 2n - 1 = 5$.

▶ **Proposition 6.** *There is an infinite family $(\mathcal{I}_n)_{n \geq 1}$ of sets of intervals with $|\mathcal{I}_n| = 3 \cdot 2^{n-1} - 2$, $\chi(\mathcal{C}[\mathcal{I}_n]) = 2n - 1$, and $\omega(\mathcal{C}[\mathcal{I}_n]) = n$.*

Proof. The construction is iterative. The family $\mathcal{I}_1$ consists of a single interval of unit length.

Now let $n > 1$ and suppose that we have defined $\mathcal{I}_{n-1}$ and want to define $\mathcal{I}_n$. We introduce two new intervals ℓ_n and r_n, both of length 3^{n-1}, that overlap slightly. Then we introduce two copies of $\mathcal{I}_{n-1}$. All intervals of one copy are contained in $\ell_n \setminus r_n$, and all intervals of the other copy are contained in $r_n \setminus \ell_n$.

The number of intervals in $\mathcal{I}_n$ is given by the recursion: $f(1) = 1$ and $f(n) = 2f(n-1)+2$, which solves to $f(n) = 3 \cdot 2^{n-1} - 2$. Furthermore, it is straightforward to observe that with each step of the construction, the size of a largest clique increases by 1.

We claim that, for $i \in [n]$, in any proper coloring of $\mathcal{C}[\mathcal{I}_i]$, the difference between the largest and the smallest color used is at least $2i - 2$. Clearly, the claim holds for $i = 1$. Now assume that it holds for $i = n - 1$. Consider any proper coloring of $\mathcal{C}[\mathcal{I}_n]$, and let m be the minimum color used in this coloring. The colors of ℓ_n and r_n must be different. Without loss of generality, suppose that the color of r_n is larger than the color of ℓ_n. In particular, the color of r_n is at least $m+1$. Now consider the copy of $\mathcal{I}_{n-1}$ contained in r_n. The color of each interval in this copy must be larger than the color of r_n, so in particular the minimum color used for this copy of $\mathcal{I}_{n-1}$ is at least $m+2$. By the inductive assumption, some interval in this copy of $\mathcal{I}_{n-1}$ receives a color that is at least $m + 2 + 2(n-1) - 2 = 2n - 2 + m$. Summing up, the difference between the largest and the smallest color used for $\mathcal{C}[\mathcal{I}_n]$ is at least $2n - 2$.

Given that the minimum color is 1, we conclude that $\chi(\mathcal{C}[\mathcal{I}_n]) \geq 2n - 1$.

For the upper bound, we color $\mathcal{C}[\mathcal{I}_n]$ as follows. For $n = 1$, we color the only interval with color 1. For $n > 1$, we color ℓ_n with color 1 and r_n with color 2. Next, for each of the two copies of $\mathcal{I}_{n-1}$, we use the proper coloring defined inductively with all colors increased by 2, see Figure 3. ◀

4 Coloring Containment Interval Graphs Is NP-Hard

In this section we show that it is NP-hard to color a containment interval graph with a given number of colors.

▶ **Theorem 7.** *Given a set $\mathcal{I}$ of intervals and a positive integer k, it is NP-hard to decide whether k colors suffice to color $\mathcal{C}[\mathcal{I}]$, that is, whether $\chi(\mathcal{C}[\mathcal{I}]) \leq k$.*

Proof. We describe a reduction from (exact) 3-SAT, i.e., the satisfiability problem where every clause contains exactly three literals. Let $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be an instance of 3-SAT where, for each clause C_i ($i \in [m]$), the literals are negated or unnegated variables from the set $\{x_1, x_2, \ldots, x_n\}$, and let $H = 5(m+1)$ be a threshold.

Using φ, we construct in polynomial time a set of intervals (with pairwise distinct endpoints) such that the corresponding containment interval graph has a proper coloring with H colors if and only if φ is satisfiable. To this end, we introduce *variable gadgets*

Figure 4 Variable gadget for the proof of Theorem 7 in its two states. The blue intervals with dots extend to the clause gadgets. The topmost blue interval (starting immediately to the right of a gray interval) indicates that x is part of a clause C_j with $j > i$; the blue interval (with a small gap) that starts immediately to the right of a red interval indicates that $\neg x$ is part of the clause C_i.

and *clause gadgets*, which are sets of intervals representing the variables and clauses of φ, respectively. Our main building structure used in these gadgets is a *Christmas tree*, that is, an ordered set of intervals where each interval contains its successor; see, for example, the set of red intervals in Figure 4. Clearly, the intervals of a Christmas tree form a totally ordered clique and any proper coloring needs to observe this order. In Figure 5, Christmas trees are represented by trapezoids. The *height* of a Christmas tree is the number of intervals it consists of.

Let $j \in [n]$. The variable gadget for x_j consists of two Christmas trees (formed by the red and gray intervals in Figure 4) whose longest intervals overlap and, for each tree, of two additional intervals (green in Figure 4). These green intervals lie immediately to the left and to the right of the shortest interval in their tree. The right green interval of the red tree overlaps the left green interval of the gray tree. Figure 4 depicts two representations of the same gadget for a variable x, each with its own coloring of the intervals (encoded by the height of the intervals; see the numbers at the right side of the gray box). The left representation with its coloring corresponds to assigning true to x, the right representation corresponds to assigning false. The height of the red tree is $H-1$ minus the number of occurrences of literals $x_{j'}$ and $\neg x_{j'}$ with $j' < j$ in φ. The height of the gray tree is that of the red tree minus the number of occurrences of $\neg x_j$ in φ. We say that x_j is set to true if the bottom interval of the gray tree has color 1; otherwise we say that x_j is set to false.

For $i \in [m]$, the gadget for clause C_i consists of a Christmas tree (light blue in Figure 5) of height $H-(5i+2) = 5(m-i)+3$. All clause gadgets are placed to the right of all variable gadgets, in the order $C_1, \ldots, C_m$ from left to right.

The key idea to transport a Boolean value from a variable gadget to a clause gadget is to add, for each occurrence of a literal $\ell_j \in \{x_j, \neg x_j\}$ in a clause C_i, an "arm" (blue intervals in Figures 4 and 5) that ends to the right of the clause gadget (the light blue Christmas tree) corresponding to C_i and starts immediately to the right of the $5i$-th interval of the gray tree (if $\ell_j = x_j$) or of the red tree (if $\ell_j = \neg x_j$) corresponding to ℓ_j. The arm is represented by a sequence of intervals that are separated by a small gap within each Christmas tree of each clause gadget passed by the arm (such that, for any two arms, their gaps are disjoint and the resulting intervals do not contain each other). Assuming that the total number of colors is H, two intervals of the same arm that are separated by a gap need to get the same color because, at the gap, $H-2$ colors are occupied by other intervals and the one remaining "wrong" color is blocked due to the green intervals of the variable gadgets; see Figures 4 and 5. The green intervals are contained by the blue intervals of the arms and need to get color H or $H-1$.

If there is a satisfying truth assignment for φ, then there is a proper coloring that colors the variable gadgets such that they represent this truth assignment. As for every clause C_i, at least one of its literals in C_i is true, the corresponding arm can use color $5i$. Then, the

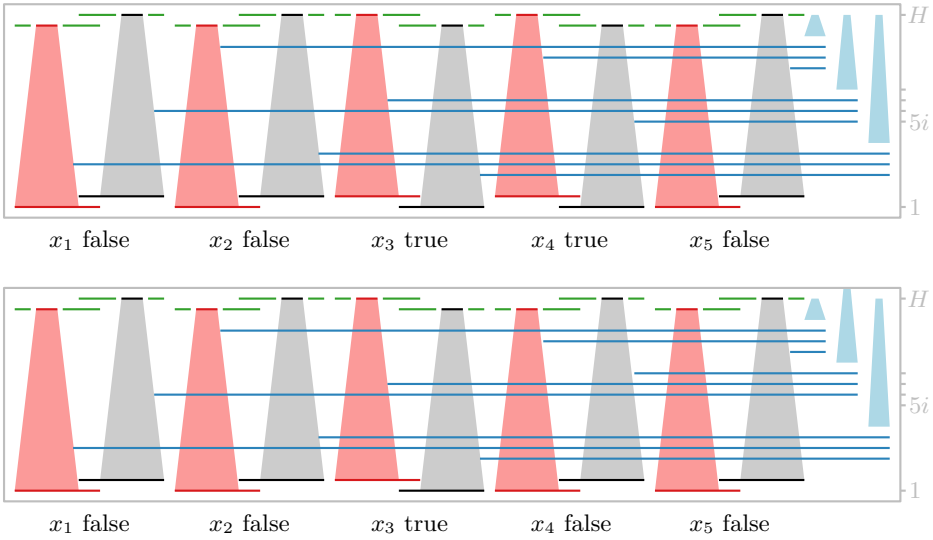


Figure 5 Variable gadgets (red and gray) and clause gadgets (blue) for the 3-SAT instance $(\neg x_2 \vee \neg x_4 \vee x_5) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3)$ with a fulfilling truth assignment (above) and a non-fulfilling assignment (below). Note that the latter uses one color more (is one level higher).

arms of the other literals that occur in C_i can use colors $5i+1$ and $5i+2$. This allows the light blue Christmas tree representing clause C_i, which has height $H-5i-2$ and is contained in the rightmost interval of each of these arms, to use the colors $\{5i+3, 5i+4, \ldots, H\}$.

Now suppose for a contradiction that there is no satisfying truth assignment for φ, but that there is a proper coloring with H colors. This coloring assigns a truth value to each variable gadget (depending on whether the bottommost interval of the red or the gray tree has color 1). Clearly, there is a clause C_i in φ that is not satisfied by this truth assignment. Hence, none of the arms connecting the clause gadget of C_i with its three corresponding variable gadgets can use color $5i$. Hence they must use colors $5i+1$, $5i+2$, and $5i+3$ (or higher). This forces the (blue) Christmas tree representing clause C_i to use colors $\{5i+4, \ldots, H, H+1\}$.

Thus, a proper coloring with H colors exists if and only if φ is satisfiable. ◀

5 Coloring Bidirectional Interval Graphs Is NP-Hard

In this section we show that it is NP-hard to color a bidirectional interval graph with a given number of colors. For a set $\mathcal{I}$ of intervals and a function o that maps every interval in $\mathcal{I}$ to an orientation (left-going or right-going), let $\mathcal{B}[\mathcal{I}, o]$ be the bidirectional interval graph induced by $\mathcal{I}$ and o.

▶ **Theorem 8.** *Given a set $\mathcal{I}$ of intervals with orientations o and a positive integer k, it is NP-hard to decide whether k colors suffice to color $\mathcal{B}[\mathcal{I}, o]$, that is, whether $\chi(\mathcal{B}[\mathcal{I}, o]) \leq k$.*

Proof. We use the same ideas as in the proof of Theorem 7, but now we reduce from MONOTONE 3-SAT, the version of 3-SAT where every clause contains only negated or only unnegated variables as literals. For an overview, see Figures 6 and 7. Let $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be the given instance of MONOTONE 3-SAT with variables $\{x_1, x_2, \ldots, x_n\}$. As before, let $H = 5(m+1)$ be the number of colors sufficient for coloring a yes-instance.

We now construct variable and clause gadgets by specifying a set $\mathcal{I}$ of intervals with orientations o. Our intervals have pairwise distinct endpoints. Our main building structures are *left- and right-going staircases*. A left-going staircase (gray in Figures 6 and 7) is an

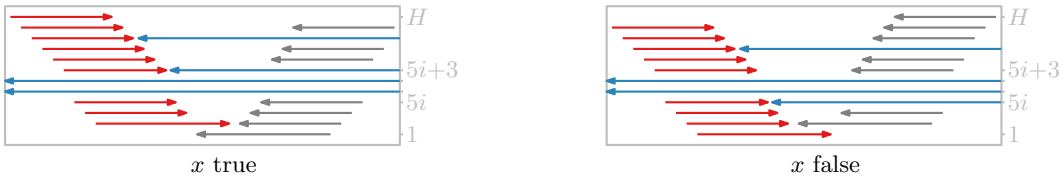


Figure 6 Variable gadget for the proof of Theorem 8 in its two states. Intervals directions are indicated by arrow heads. The blue intervals extend (to the right) to the clause gadgets of only negated variables. The two blue arrow heads starting next to the red intervals indicate that the clause C_i and a clause C_j with $j > i$ contain the literal $\neg x$.

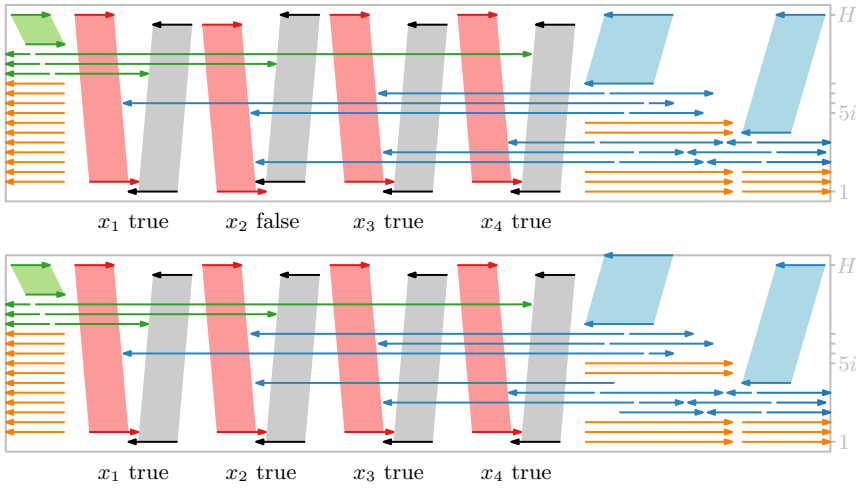


Figure 7 Variable gadgets (red and gray) and clause gadgets (light green and light blue) for the Monotone 3-Sat instance $(x_1 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$ with a fulfilling truth assignment (top) and a non-fulfilling assignment (bottom), which use H and $H + 1$ colors, respectively. Interval directions are indicated by arrow heads.

ordered set of left-going intervals that share a common point and whose left and right endpoints are in the order of the set. A right-going staircase is symmetric (red in Figures 6 and 7). Observe that staircases in (bi)directional interval graphs behave like Christmas trees in containment graphs: they form totally ordered cliques. The *height* of a staircase is the number of its intervals. In Figure 7, we draw staircases as parallelograms and we indicate left- and right-going intervals by arrow heads.

Let $i \in [m]$. If the clause C_i has only negated literals, we again have three "arms" in C_i starting to the right of the $5i$-th interval of the red staircase of the three corresponding variables; see the blue intervals in Figures 6 and 7. The intervals of these arms are left-going and they are not split by a gap at the other variable gadgets this time because now we need to contain the left-going intervals of the staircases to have edges instead of arcs in $\mathcal{B}[\mathcal{I}, o]$. The arms end below a left-going light blue staircase (see Figure 7 on the upper right side) of height $5(m - i) + 3$ whose maximum color is H if and only if none of the corresponding arms gets a color greater than $5i + 2$.

Note that we need to avoid arcs between the arms. Therefore, we let every arm have a gap below each blue staircase that it passes. These gaps do not overlap and their order is inverse to the order of the left endpoints of the involved intervals. We continue the arm with a right-going interval (to avoid an arc with the blue staircase and the other arms) ending to

the right of the blue staircase, where we continue again with a left-going interval. Consider such an arm with intervals I and I' (going in different directions) around a gap. At this gap, it is important that no color smaller than the color of I is available for I', forcing I' to also get the color of I. Hence, we add long left-going intervals blocking every color not occupied by an arm or the blue staircase; see the orange intervals in Figure 7.

For every clause with only unnegated literals, we use the same construction but mirrored (connecting to the gray staircases); see the green intervals and light green staircases depicted in the top left corner of Figure 7.

We now have the same conditions as in the proof of Theorem 7: If there is a satisfying truth assignment for φ, there is a proper coloring, which colors the variable gadgets such that they represent this truth assignment. Again, for every $i \in [m]$, clause C_i contains at least one literal set to true. Hence, the corresponding arm can get color $5i$, and the other two arms and the (light blue or light green) staircase of C_i get colors $\{5i+1, 5i+2, \ldots, H\}$.

If there is no satisfying truth assignment for φ, then there is a clause C_i none of whose arms (as a whole) can get color $5i$. If each arm occupies only one layer, then an interval of the clause gadget of C_i requires color $H+1$. If there is an arm occupying more than one layer, then below a clause gadget of some clause $C_{i'}$, there are two colors blocked by this arm (at one of its gaps). Then, however, an interval of the clique of size $H-1$ at this gap belonging to the (light blue or light green) staircase of $C_{i'}$, to the other arms, or to the orange "blocker" intervals requires color $H+1$; see Figure 7 for such an example. ◀

6 Coloring General Mixed Interval Graphs

In this section we consider a further generalization of mixed interval graphs. We are dealing with an interval graph G whose edges can be arbitrarily oriented (or stay undirected). In other words, the edge directions are not related to the geometry of the intervals.

Observe that a proper coloring of G exists if and only if G does not contain a directed cycle. Let $\chi(G)$ denote the minimum number of colors in a proper coloring of G, if it exists, or ∞ otherwise. We point out that the existence of a directed cycle can be determined in polynomial time (using, for example, depth-first search).

Note that clearly we have $\omega(G) \leq \chi(G)$. However, there is another parameter that enforces a large chromatic number even in sparse graphs. A *directed path* (of length t) in G is a sequence of vertices $\langle v_1, v_2, \ldots, v_{t+1}\rangle$, such that, for each $i \in [t]$, the arc (v_i, v_{i+1}) exists. Let $\lambda(G)$ denote the length of a longest directed path in G.

Note that the vertices in a directed path receive pairwise distinct colors in any proper coloring. Thus we have $\chi(G) \geq \lambda(G)+1$, and consequently $\chi(G) \geq \max\{\omega(G), \lambda(G)+1\}$.

▶ **Theorem 9.** *Let G be a mixed interval graph without directed cycles. Then $\chi(G) \leq (\lambda(G)+1)\cdot\omega(G)$.*

Proof. Let V denote the vertex set of G. Let $G^{\rightarrow}$ be the graph obtained from G by removing all edges. Clearly, $G^{\rightarrow}$ is a DAG. We partition V into *layers* $L_0, L_1, \ldots$ as follows. The set L_0 consists of the vertices that are sources in $G^{\rightarrow}$, i.e., they do not have incoming arcs. Then, for $i = 1, 2, \ldots$, we iteratively define L_i to be the set of sources in $G^{\rightarrow} \setminus \bigcup_{j=0}^{i-1} L_j$. Note that $\lambda(G) = \max\{i \colon L_i \neq \emptyset\}$. For $x \in V$, let $\ell(x) \in \{0, \ldots, \lambda(G)\}$ denote the unique i such that $x \in L_i$.

Recall that the underlying undirected graph of G, $U(G)$, is an (undirected) interval graph, and thus $\chi(U(G)) = \omega(U(G)) = \omega(G)$. Let $c \colon V \to [\omega(U(G))]$ be an optimal proper coloring of $U(G)$.

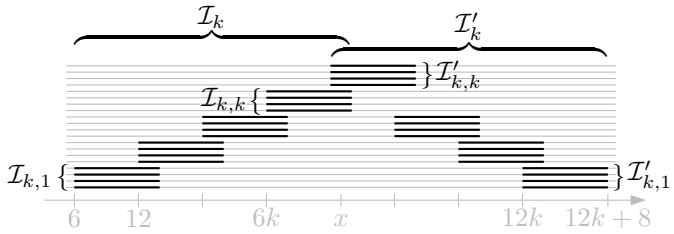


Figure 8 For any $k \geq 1$, the set $\mathcal{I}_k \cup \mathcal{I}'_k$ of intervals gives rise to a mixed interval graph G_k with $2k^2$ vertices, $\lambda(G_k) = k-1$, $\omega(G_k) = 2k$, and $\chi(G_k) = (k+1) \cdot k = (\lambda(G_k)+2) \cdot \omega(G_k)/2$.

Now we define a coloring f of G: for $x \in V$, let $f(x) = \ell(x) \cdot \omega(G) + c(x)$. Note that $1 \leq f(x) \leq (\lambda(G)+1) \cdot \omega(G)$. We claim that f is a proper coloring.

Consider an edge $\{x, y\}$. As this is also an edge in $U(G)$, we obtain that $c(x) \neq c(y)$, and so $f(x) \neq f(y)$. Now consider an arc (x, y). Its existence implies that $\ell(x) < \ell(y)$, and thus $f(x) < f(y)$. ◀

For some instances, the above bound is asymptotically tight.

▶ **Proposition 10.** *There is an infinite family $(G_k)_{k\geq 1}$ of mixed interval graphs with $|V(G_k)| = 2k^2$, $\lambda(G_k) = k-1$, $\omega(G_k) = 2k$, and $\chi(G_k) = (k+1) \cdot k = (\lambda(G_k)+2) \cdot \omega(G_k)/2$.*

Proof. Let $\mathcal{I}_k = \mathcal{I}_{k,1} \cup \mathcal{I}_{k,2} \cup \cdots \cup \mathcal{I}_{k,k}$ be a set of k^2 intervals defined as follows; see Figure 8 for $\mathcal{I}_4$. For $i \in [k]$, let $\mathcal{I}_{k,i}$ be a multiset that contains k copies of the interval $[6i, 6i+8]$. Similarly, let $\mathcal{I}'_k = \mathcal{I}'_{k,1} \cup \mathcal{I}'_{k,2} \cup \cdots \cup \mathcal{I}'_{k,k}$ be a set of k^2 intervals such that $\mathcal{I}'_k$ is the image of mirroring $\mathcal{I}_k$ at the point $x = 6k+7$. Note that, for $i \in [k-1]$, every interval in $\mathcal{I}_{k,i}$ intersects every interval in $\mathcal{I}_{k,i+1}$ and every interval in $\mathcal{I}'_{k,i}$ intersects every interval in $\mathcal{I}'_{k,i+1}$. Additionally, every interval in $\mathcal{I}_{k,k}$ intersects every interval in $\mathcal{I}'_{k,k}$.

Let G_k be a mixed interval graph for the set $\mathcal{I}_k \cup \mathcal{I}'_k$. We direct the edges of G_k as follows. Let $\{I, I'\}$ be a pair of intervals in $\mathcal{I}_k \cup \mathcal{I}'_k$ that intersect each other. If I and I' are copies of the same interval, then $\{I, I'\}$ is an edge of G_k. Otherwise, (I, I') is an arc of G_k if $\{I, I'\} \subseteq \mathcal{I}_k$ and I lies further to the left than I', if $\{I, I'\} \subseteq \mathcal{I}'_k$ and I lies further to the right than I', or if $(I, I') \in \mathcal{I}_{k,k} \times \mathcal{I}'_{k,k}$. It is easy to see that G_k has the desired properties. ◀

Note that the mixed interval graphs that we constructed in the proof above are even *directional* interval graphs.

7 Open Problems

The obvious open problems are improvements to the results in Table 1, in particular: Is there a constant-factor approximation algorithm for coloring general mixed interval graphs? For applications in graph drawing, a better-than-2 approximation for coloring bidirectional interval graphs is of particular interest.

Is there a linear-time recognition algorithm for directional or containment interval graphs? Is there a polynomial-time recognition algorithm for bidirectional interval graphs?

Using a reduction from Max-3-SAT instead of 3-SAT, it may be possible to adjust our NP-hardness proofs in order to show APX-hardness. To this end, the difference in the number of colors needed to color a yes-instance and the number of colors needed to color a no-instance would have to be proportional to the number of clauses that cannot be satisfied. We were not able to force such a large difference, hence we leave the APX-hardness of (or the existence of a PTAS for) coloring containment and birectional interval graphs open.

References

1 Matthias Beck, Daniel Blado, Joseph Crawford, Taïna Jean-Louis, and Michael Young. Mixed graph colorings. In *Proc. Sci. Nat. Conf. of the Society for Advancement of Hispanics/Chicanos and Native Americans*, 2012.

2 Matthias Beck, Daniel Blado, Joseph Crawford, Taïna Jean-Louis, and Michael Young. On weak chromatic polynomials of mixed graphs. *Graphs Combin.*, 31:91–98, 2015. doi:10.1007/s00373-013-1381-1.

3 Peter Brucker. *Scheduling Algorithms*. Springer, 5 edition, 1995. doi:10.1007/978-3-540-69516-5.

4 Dominique de Werra. Restricted coloring models for timetabling. *Discrete Math.*, 165–166:161–170, 1997. doi:10.1016/S0012-365X(96)00208-7.

5 Hanna Furmańczyk, Adrian Kosowski, and Paweł Żyliński. A note on mixed tree coloring. *Inf. Process. Lett.*, 106(4):133–135, 2008. doi:10.1016/j.ipl.2007.11.003.

6 Hanna Furmańczyk, Adrian Kosowski, and Paweł Żyliński. Scheduling with precedence constraints: Mixed graph coloring in series-parallel graphs. In *Proc. PPAM'07*, pages 1001–1008, 2008. doi:10.1007/978-3-540-68111-3_106.

7 Grzegorz Gutowski, Konstanty Junosza-Szaniawski, Felix Klesen, Paweł Rzążewski, Alexander Wolff, and Johannes Zink. Coloring and recognizing directed interval graphs. ArXiv report, 2023. doi:10.48550/arXiv.2303.07960.

8 Grzegorz Gutowski, Florian Mittelstädt, Ignaz Rutter, Joachim Spoerhase, Alexander Wolff, and Johannes Zink. Coloring mixed and directional interval graphs. In Patrizio Angelini and Reinhard von Hanxleden, editors, *Proc. 30th Int. Symp. Graph Drawing & Network Vis. (GD'22)*, volume 13764 of *LNCS*, pages 418–431. Springer, 2023. doi:10.1007/978-3-031-22203-0_30.

9 Pierre Hansen, Julio Kuplinsky, and Dominique de Werra. Mixed graph colorings. *Math. Methods Oper. Res.*, 45:145–160, 1997. doi:10.1007/BF01194253.

10 George S. Lueker and Kellogg S. Booth. A linear time algorithm for deciding interval graph isomorphism. *J. ACM*, 26(2):183–195, 1979. doi:10.1145/322123.322125.

11 Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discrete Math.*, 201(1):189–241, 1999. doi:10.1016/S0012-365X(98)00319-7.

12 Bernard Ries and Dominique de Werra. On two coloring problems in mixed graphs. *Eur. J. Comb.*, 29(3):712–725, 2008. doi:10.1016/j.ejc.2007.03.006.

13 Paolo Serafini and Walter Ukovich. A mathematical model for the fixed-time traffic control problem. *Europ. J. Oper. Res.*, 42(2):152–165, 1989. doi:10.1016/0377-2217(89)90318-4.

14 Yuri N. Sotskov. Mixed graph colorings: A historical review. *Mathematics*, 8(3):385:1–24, 2020. doi:10.3390/math8030385.

15 Yuri N. Sotskov, Vjacheslav S. Tanaev, and Frank Werner. Scheduling problems and mixed graph colorings. *Optimization*, 51(3):597–624, 2002. doi:10.1080/0233193021000004994.

16 Yuri N. Sotskov and Vyacheslav S. Tanaev. Chromatic polynomial of a mixed graph. *Vestsi Akademii Navuk BSSR. Seryya Fizika-Matematychnykh Navuk*, 6:20–23, 1976.

17 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.

18 Vjacheslav S. Tanaev, Yuri N. Sotskov, and V.A. Strusevich. *Scheduling Theory: Multi-Stage Systems*. Kluwer Academic Publishers, 1994.

19 Johannes Zink, Julian Walter, Joachim Baumeister, and Alexander Wolff. Layered drawing of undirected graphs with generalized port constraints. *Comput. Geom.*, 105–106(101886):1–29, 2022. doi:10.1016/j.comgeo.2022.101886.

Shortest Beer Path Queries Based on Graph Decomposition

Tesshu Hanaka
Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan

Hirotaka Ono
Graduate School of Informatics, Nagoya University, Japan

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

Kosuke Sugiyama
Graduate School of Informatics, Nagoya University, Japan

Abstract

Given a directed edge-weighted graph $G = (V, E)$ with beer vertices $B \subseteq V$, a beer path between two vertices u and v is a path between u and v that visits at least one beer vertex in B, and the beer distance between two vertices is the shortest length of beer paths. We consider *indexing problems* on beer paths, that is, a graph is given a priori, and we construct some data structures (called indexes) for the graph. Then later, we are given two vertices, and we find the beer distance or beer path between them using the data structure. For such a scheme, efficient algorithms using indexes for the beer distance and beer path queries have been proposed for outerplanar graphs and interval graphs. For example, Bacic et al. (2021) present indexes with size $O(n)$ for outerplanar graphs and an algorithm using them that answers the beer distance between given two vertices in $O(\alpha(n))$ time, where $\alpha(\cdot)$ is the inverse Ackermann function; the performance is shown to be optimal. This paper proposes indexing data structures and algorithms for beer path queries on general graphs based on two types of graph decomposition: the tree decomposition and the triconnected component decomposition. We propose indexes with size $O(m + nr^2)$ based on the triconnected component decomposition, where r is the size of the largest triconnected component. For a given query $u, v \in V$, our algorithm using the indexes can output the beer distance in query time $O(\alpha(m))$. In particular, our indexing data structures and algorithms achieve the optimal performance (the space and the query time) for series-parallel graphs, which is a wider class of outerplanar graphs.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases graph algorithm, shortest path problem, SPQR tree

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.37

Related Version *Extended Version*: `https://arxiv.org/abs/2307.02787`

Funding *Tesshu Hanaka*: JSPS KAKENHI Grant Numbers JP21H05852, JP21K17707, JP22H00513, and JP23H04388
Hirotaka Ono: JSPS KAKENHI Grant Numbers JP20H00081, JP20H05967, JP21K19765, JP22H00513
Kunihiko Sadakane: JSPS KAKENHI Grant Number JP20H05967

1 Introduction

Given a directed edge-weighted graph $G = (V, E)$ with beer vertices $B \subseteq V$, a beer path between two vertices u and v is a path between u and v that visits at least one beer vertex in B, and the beer distance between two vertices is the shortest length of beer paths. Here, a graph with B, the set of beer stores, is called a beer graph. The names "beer path" and "beer distance" come from the following story: A person will visit a friend but does not want

to show up empty-handed, and they decide to pick up some beer along the way. They would like to take the fastest way to go from their place to their friend's place while stopping at a beer store to buy some drinks.

The notion of the beer path was recently introduced by Bacic et al. [1]. Although the name is somewhat like a fable, we often encounter similar situations as the above story. Instead of beer stores, we want to stop at a gas station along the way, for example.

Just computing the beer distance or a beer path with the beer distance is easy. A beer path with the beer distance always consists of two shortest paths: from the source to one of the beer stores and from the beer store to the destination. We can therefore compute them by solving the single source shortest path problem twice from the source and from the destination, and taking the minimum beer vertex among B.

We consider *indexing problems* on beer paths, that is, a graph is given a priori, and we construct some data structures (called indexes) for the graph. Then later, we are given two vertices, and we find the beer distance or beer path between them using the data structure. This is more efficient than algorithms without using any indexes if we need to solve queries for many pairs of vertices. Indeed, car navigation systems might equip such indexing mechanisms; since a system has map information as a graph in advance, it can make indexed information by preprocessing, which enables it to quickly output candidates of reasonable routes from the current position as soon as receiving a goal point. Such a scenario is helpful also for the beer path setting. Efficient algorithms using indexes for the beer distance and beer path queries have been proposed for outerplanar graphs [1] and interval graphs [5].

This paper presents indexes with efficient query algorithms using graph decomposition for more general classes of graphs. Namely, we consider graphs of bounded treewidth [3] and graphs with bounded triconnected components size [9]. The performance of our indexing and algorithm generalizes that for outerplanar graphs in [1]; if we apply our indexing and algorithm for outerplanar graphs, the space for indexes, preprocessing time, and query time are equivalent to those of [1]. Furthermore, ours can be applied for general graphs, though the performance worsens for graphs of large treewidth and with a large triconnected component.

1.1 Related Work

For undirected outerplanar beer graphs G of order n, Bacic et al. [1] present indexes with size $O(n)$, which can be preprocessed in $O(n)$ time. For any two query vertices u and v, (i) the beer distance between u and v can be reported in $O(\alpha(n))$ time, where $\alpha(n)$ is the inverse Ackermann function, and (ii) a beer path with the beer distance between u and v can be reported in $O(L)$ time, where L is the number of vertices on this path. The query time is shown to be optimal.

For unweighted interval graphs with beer vertices B, Das et al. [5] provides a representation using $2n \log n + O(n) + O(|B| \log n)$ bits. This data structure answers beer distance queries in $O(\log^{\varepsilon} n)$ time for any constant $\varepsilon > 0$ and shortest beer path queries in $O(\log^{\varepsilon} n + L)$ time. They also present a trade-off relation between space and query time. These results are summarized in Table 1.

Other than the beer path problem, Farzan and Kamali [6] proposed a distance oracle for graphs with n vertices and treewidth k using asymptotically optimal $k(n + o(n) - k/2) + O(n)$ bit space which can answer a query in $O(k^3 \log^3 k)$ time. For general graphs, indexes for shortest paths (i.e., distance queries) and max flow queries based on the triconnected component decomposition have been proposed [10].

Table 1 Comparison of results.

Graphs		Preprocessing Complexity		Query time
		Space (words)	Time	
Outerplanar graphs [1] (undirected, $W = \mathbb{R}_{\geq 0}$)		$O(m)$ $(= O(n))$		$O(\alpha(n))$
Interval graphs [5] (undirected, unweighted)		$O(n + \lvert B\rvert)$	$O(n + \lvert B\rvert)^*$	$O(\log^{\varepsilon} n)$
		$O(n + \lvert B\rvert \log\log n)$	$O(n + \lvert B\rvert \log\log n)^*$	$O(\log\log n)$
Ours	tri. decomp. (undirected, $W = \mathbb{Z}_{\geq 0}$)	$O(m)$	$O(m + nr^3)$	$O(r^2 + \alpha(m))$
		$O(m + nr^2)$	$O(m + nr^4)$	$O(\alpha(m))$
	tri. decomp. ($W = \mathbb{R}_{\geq 0}$)	$O(m)$	$O((m + n\log r_+)r_+^2)$	$O(r^2 \log r_+ + \alpha(m))$
		$O(m + nr^2)$	$O((m + n\log r_+)r_+^3)$	$O(\alpha(m))$
	tree decomp. ($W = \mathbb{R}_{\geq 0}$)	$O(t^3 n)$	$O(t^8 n)$	$O(t^7 + \alpha(tn))$
		$O(t^5 n)$	$O(t^{10} n)$	$O(t^6 + \alpha(tn))$

n: the number of vertices
m: the number of edges
r: the size of maximum triconnected components, $r_+ = \max\{1, r\}$
t: the treewidth
$\alpha(\cdot)$: the inverse Ackermann function
*: not explicitly mentioned

1.2 Our Contribution

We present indexing data structures and query algorithms for beer distance and beer path queries for general graphs based on graph decomposition. As graph decomposition, we use the tree decomposition [3] and the triconnected component decomposition [9]. The obtained results are summarized in Table 1.

We first present faster query algorithms using properties of the triconnected component decomposition. In this approach, we use r, the size of the largest triconnected component in a graph, as the parameter to evaluate the efficiency of algorithms. Note that r is not the number of edges in the largest triconnected component; it is the number of edges in a component after contracting every biconnected component into an edge and therefore it is not so large in practice. The formal definition will be given in Section 2.2. Our data structure uses $O(m + r \cdot \min\{m, rn\})$ space, and the algorithm for undirected graphs with nonnegative integer edge weights requires $O(m + r^3 \cdot \min\{m, rn\})$ time for preprocessing, and it answers for each query in $O(\alpha(m))$ time. For directed graphs with nonnegative edge weights, the preprocessing time and query time are, respectively $O(m + r^3(m + n\log r_+))$ and $O(\alpha(m))$. Since the size of indexes and query time have a trade-off relation, a little slower query time can achieve an indexing data structure with less memory. In such a scenario, another data structure uses $O(m)$ space, and the algorithm for undirected graphs with nonnegative integer edge weights requires $O(m + r^2 \cdot \min\{m, rn\})$ time for preprocessing, and it answers for each query in $O(r^2 + \alpha(m))$ time. For directed graphs with nonnegative edge weights, the preprocessing time and query time are, respectively $O(m + r^2(m + n\log r_+))$ and $O(r^2 \log r_+ + \alpha(m))$.

Because triconnected component decomposition can be regarded as a tree decomposition, we extend our query algorithms for graphs represented by using the tree decomposition. Though computing the exact treewidth is NP-hard, whereas triconnected component decomposition is done in linear time [8], and query time complexities using tree decomposition is

larger than using triconnected component decomposition, the treewidth is always at most r and therefore algorithms based on the tree decomposition are faster in some cases. In view of these, we remake the indexing data structures and algorithms for tree decomposition. The indexing data structure requires $O(t^5 n)$ space and $O(t^{10} n)$ time to construct, and the algorithm can answer a query in $O(t^6 + \alpha(tn))$ time.

Note that for series-parallel graphs $r = 0$, $t = 2$, and $m = O(n)$ hold. This implies that for series-parallel graphs, our indexing data structures use $O(n)$ space, and the algorithms can answer each query in $O(\alpha(n))$ time. Since the class of series-parallel graphs is a super class of outerplanar graphs, our results fairly extend the optimal result for outerplanar graphs by [1].

The rest of the paper is organized as follows. Section 2 is for preliminaries. Sections 3 and 4 present the main parts that describe the indexing and algorithms under triconnected decomposition. Section B shows how we remake that to those under tree decomposition.

2 Preliminaries

Let $\mathbb{Z}_{\geq 0}$ be the set of nonnegative integers and $\mathbb{R}_{\geq 0}$ be the set of nonnegative real numbers. For nonnegative integers $i, j \in \mathbb{Z}_{\geq 0}$ $(i \leq j)$, let $[i, j] = \{i, i+1, \ldots, j-1, j\}$.

For a graph G, let $V(G)$ and $E(G)$ denote its vertex and edge sets, respectively. For two graphs G and G', let $G \setminus G' = (V(G) \setminus V(G'), E(G) \setminus E(G'))$ and $G \cup G' = (V(G) \cup V(G'), E(G) \cup E(G'))$. Also, for a graph G and a set of vertex pairs $F \subseteq V(G) \times V(G)$, let $G \setminus F = (V(G), E(G) \setminus F)$ and $G \cup F = (V(G), E(G) \cup F)$. Furthermore, for a graph G and its vertex subset $S \subseteq V(G)$, let $G[S]$ be the subgraph of G induced by S.

2.1 Shortest Path Problem and Beer Path Problem / Query

Suppose we are given a graph G, an edge weight $w\colon E(G) \to W$, and a vertex subset $B \subseteq V(G)$. Note that in this paper, we assume $W = \mathbb{Z}_{\geq 0}$ or $W = \mathbb{R}_{\geq 0}$.

For vertices $u, v \in V(G)$, a path from u to v in G is called a *u-v path* in G. Usually, a u-v path is not unique. The length of a path is defined by the sum of the edge weights on the path. The shortest length of all u-v paths is called the u-v distance in G, denoted by $\mathrm{d}((G, w), u, v)$. Also, for vertices $u, v \in V(G)$, a walk from u to v passing through a vertex belonging to B at least once is called a *u-v beer path* in G. The length of a u-v beer path is similarly defined as the length of a u-v path, u-v beer distance in G is defined by the shortest length of all u-v beer paths and is denoted by $\mathrm{d}^{\mathrm{B}}((G, w, B), u, v)$. Note that if w and B are clear from the context, we omit them and denote $\mathrm{d}((G, w), u, v)$ as $\mathrm{d}(G, u, v)$ and $\mathrm{d}^{\mathrm{B}}((G, w, B), u, v)$ as $\mathrm{d}^{\mathrm{B}}(G, u, v)$. Then, a vector whose elements are the distance and the beer distance is denoted by

$$\vec{d}(G, u, v) = \begin{pmatrix} \mathrm{d}(G, u, v) \\ \mathrm{d}^{\mathrm{B}}(G, u, v) \end{pmatrix}.$$

For a given G, w, B and vertices $u, v \in V(G)$, the problem of finding $\mathrm{d}(G, u, v)$ (or one u-v path that realizes it) is called Shortest Path and the problem of finding $\mathrm{d}^{\mathrm{B}}(G, u, v)$ (or one u-v beer path that realizes it) is called Beer Path. For given u and v, the query asked to return $\mathrm{d}^{\mathrm{B}}(G, u, v)$ or one u-v beer path with length $\mathrm{d}^{\mathrm{B}}(G, u, v)$ is called *Beer Path Query* on G, w, B.

Here, we review algorithms for the shortest path problem and their computational complexity. When $W = \mathbb{Z}_{\geq 0}$ and G is an undirected graph, the shortest path problem can be solved in $O(m)$ time by using Thorup's algorithm [11]. When $W = \mathbb{R}_{\geq 0}$, the shortest

path problem can be solved in $O(m + n \log n)$ time by Dijkstra's algorithm using Fibonacci heap [7]. Hereafter, let $\mathrm{ALG}(G)$ denote the computational time to solve Shortest Path Problem by one of the above algorithms according to the setting; for example, if G is undirected and $W = \mathbb{Z}_{\geq 0}$, $\mathrm{ALG}(G) = O(m)$, and if $W = \mathbb{R}_{\geq 0}$, $\mathrm{ALG}(G) = O(m + n \log m)$.

2.2 SPQR tree

Let G be a biconnected (multi) undirected graph and $\{u, v\}$ be its vertex pair. If $G[V(G) \setminus \{u, v\}]$ is disconnected or u and v are adjacent in G, $\{u, v\}$ is called a split pair of G. We denote the set of split pairs of G by Spl_G. For $\{u, v\} \in \mathrm{Spl}_G$, a maximal subgraph H of G satisfying $\{u, v\} \notin \mathrm{Spl}_H$, and the graph $(\{u, v\}, \{e\})$ consisting of the edge e connecting u and v, are called a split component of the split pair $\{u, v\}$ of G. We denote the set of split components of the split pair $\{u, v\}$ of G by $\mathrm{SplCom}_G(u, v)$. For $\{u, v\} \in \mathrm{Spl}_G$ and $\{s, t\} \in E(G)$, we say that $\{u, v\}$ is maximal with respect to $\{s, t\}$ if vertices u, v, s, t are in the same split component for any split pair $\{u', v'\}$.

For example, for the graph G shown in Figure 6, $\mathrm{Spl}_G = E(G) \cup \{\{1, 6\}, \{1, 7\}, \{2, 6\}, \{4, 6\}\}$ and $\mathrm{SplCom}_G(1, 6) = \{G_1, G_2, G_3\}$, $\mathrm{SplCom}_G(2, 6) = \{H_1, H_2, H_3\}$. Also, $\{1, 6\} \in \mathrm{Spl}_G$ is maximal with respect to the edge $\{1, 2\}$. Furthermore, $\{1, 6\} \in \mathrm{Spl}_G$ is not maximal with respect to the edge $\{2, 5\}$ because no component in $\mathrm{SplCom}_G(2, 6)$ containing all vertices 1, 6, 2, and 5.

For an edge $e = \{u, v\} \in E(G)$, we define an SPQR tree $\mathcal{T}(G, e)$ of G. Here, e is called a *reference edge* of $\mathcal{T}(G, e)$ of G. Each node μ of $\mathcal{T}(G, e)$ is associated with a graph Sk_μ. The root node of $\mathcal{T}(G, e)$ is denoted by μ_e. The $\mathcal{T}(G, e)$ is defined recursively as follows.

Trivial Case If $\mathrm{SplCom}_G(u, v) = \{(\{u, v\}, \{e\}), (\{u, v\}, \{e'\})\}$ $(e' \in E(G))$, that is, G is a two vertices multi graph consisting of two edges e, e', $\mathcal{T}(G, e) = (\{\mu_e\}, \emptyset)$, $\mathrm{Sk}_{\mu_e} = G$. Also, μ_e is said to be a Q node.

Series Case Let $\mathrm{SplCom}_G(u, v) = \{(\{u, v\}, \{e\}), H\}$, where H is formed by a series connection of $k(\geq 2)$ connected components $H_1, \ldots H_k$. Then, for vertices $u = c_0, c_1, c_2, \ldots, c_{k-1}, c_k = v$ ($c_1, \ldots, c_{k-1}$ are cut vertices of G), let c_{i-1}, c_i be the only vertices belonging to H_i $(1 \leq i \leq k)$. In this case, if $e_i = \{c_{i-1}, c_i\}$ $(1 \leq i \leq k)$, then

$$\mathcal{T}(G, e) = (\{\mu_e\}, \emptyset) \cup \bigcup_{1 \leq i \leq k} \left(\mathcal{T}(H_i \cup \{e_i\}, e_i) \cup \{\{\mu_e, \mu_{e_i}\}\}\right),$$

$$\mathrm{Sk}_{\mu_e} = (\{c_0, \ldots, c_k\}, \{e, e_1, \ldots, e_k\}).$$

Also, μ_e is said to be an S node.

Parallel Case If $\mathrm{SplCom}_G(u, v) = \{(\{u, v\}, \{e\}), H_1, \ldots, H_k\}$ $(k \geq 2)$, that is, G is formed by the parallel connection of 3 or more split components of $\{u, v\}$. In this case, if we let e_i denote the edge corresponding to H_i, then $\mathrm{Sk}_{\mu_e} = (\{u, v\}, \{e, e_1, \ldots, e_k\})$ and $\mathcal{T}(G, e)$ is defined as same as series case. Also, μ_e is said to be a P node.

Rigid Case If the above does not apply, that is, $\mathrm{SplCom}_G(u, v) = \{(\{u, v\}, \{e\}), H\}$ and H has no cut vertices, let all maximal split pairs for $\{u, v\}$ in $\mathrm{Spl}_G \setminus \{\{u, v\}\}$ be $\{u_i, v_i\}$ $(1 \leq i \leq k, k \geq 1)$. Also, for each i, let H_i be the union of the split components for $\{u_i, v_i\}$ that does not contain e. That is, $H_i = \bigcup_{H \in \mathrm{SplCom}_G(u_i, v_i): e \notin E(H)} H$. In this case, if we let e_i denote the edge corresponding to H_i $(1 \leq i \leq k)$, then

$$\mathrm{Sk}_{\mu_e} = \left(\{u, v\} \cup \bigcup_{1 \leq i \leq k} \{u_i, v_i\}, \{e\} \cup \bigcup_{1 \leq i \leq k} \{e_i\}\right)$$

and $\mathcal{T}(G, e)$ is defined as same as series case. Also, μ_e is said to be an R node.

The tree $(\{\rho, \emptyset\}) \cup \mathcal{T}(G, e) \cup \{\{\rho, \mu_e\}\}$ obtained by connecting the tree $\mathcal{T}(G, e)$ obtained by the above definition and Q node ρ with the graph $\mathrm{Sk}_\rho = (\{u, v\}, \{e\})$ as a root is called the SPQR tree of G with respect to edge e. Hereafter, we simply call it an SPQR tree and denote it by $\mathcal{T}$. Also, we denote the only child node μ_e of the root node ρ by ρ'.

For each node $\mu \in V(\mathcal{T}) \setminus \{\rho\}$ of $\mathcal{T}$, each edge of Sk_μ is a skeleton of a certain graph, so Sk_μ is called the skeleton graph of μ. Let $n_\mu = |V(\mathrm{Sk}_\mu)|$ be the number of vertices and $m_\mu = |E(\mathrm{Sk}_\mu)|$ be the number of edges of the skeleton Sk_μ. Also, let the reference edge of μ be $\mathrm{Ref}_\mu = \{x_\mu, y_\mu\}$ and let Ch_μ and Des_μ be the sets of child and descendant nodes of μ in $\mathcal{T}$, respectively. We denote the set consisting of S, P, Q, and R nodes by $S_\mathcal{T}, P_\mathcal{T}, Q_\mathcal{T}, R_\mathcal{T}$, respectively. For each $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, let G_μ be the subgraph of G corresponding to the graph of Sk_μ without the reference edge. This can be expressed as $G_\mu = (\{x_\mu, y_\mu\}, \{\{x_\mu, y_\mu\}\})$ if $\mu \in Q_\mathcal{T}$, otherwise $G_\mu = \bigcup_{\lambda \in \mathrm{Ch}_\mu} G_\lambda$. An example of a SPQR tree is shown in Figure 1.

The following is known for the SPQR tree $\mathcal{T}$ of G.

▶ **Lemma 1.** *Let G be a biconnected undirected graph with n vertices and m edges, and $\mathcal{T}$ be its SPQR tree. For each node $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, $\{x_\mu, y_\mu\} \in \mathrm{Spl}_G$. If $\mu \in R_\mathcal{T}$, Sk_μ is a triconnected graph. Also, $|Q_\mathcal{T}| = m$, $|S_\mathcal{T} \cup P_\mathcal{T} \cup R_\mathcal{T}| = O(n)$, $\sum_{\mu \in S_\mathcal{T} \cup P_\mathcal{T} \cup R_\mathcal{T}} m_\mu = O(m)$, and $\sum_{\mu \in V(\mathcal{T})} n_\mu = O(n)$ hold. Furthermore, $\mathcal{T}$ van be computed in $O(n + m)$ time.*

Let $r = \max_{\mu \in R_\mathcal{T}}\{m_\mu\}$ be the maximum number of edges in the skeleton of the R node (triconnected graph). Note that if $R_\mathcal{T} = \emptyset$), $r = 0$. Also, $r_+ = \max\{1, r\}$.

An SPQR tree for a directed graph is defined as a graph whose skeleton is replaced by a directed graph after computing the SPQR tree by considering the graph as an undirected graph. In this case, for each $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, we consider two reference edges $\langle x_\mu, y_\mu \rangle, \langle y_\mu, x_\mu \rangle$ and let $\mathrm{Ref}_\mu = \{\langle x_\mu, y_\mu \rangle, \langle y_\mu, x_\mu \rangle\}$.

2.3 Query Problems

We present all query problems that will be used in later.

Range Minimum Query For a given array $(a) = a[1], a[2], \ldots, a[n]$ of length n, the query defined by the following pair of inputs and outputs is called Range Minimum Query for the array (a).

Input Positive integers i, j $(1 \leq i \leq j \leq n)$,

Output Minimum value in the subarray $a[i], a[i+1], \ldots, a[j-1], a[j]$ of the array (a).

This query can be answered in $O(1)$ time by preprocessing in $O(n)$ space and $O(n)$ time [2].

Lowest Common Ancestor Query Given a rooted tree T with n vertices. The query defined by the following pair of input and output is called the lowest common ancestor query for the rooted tree T.

Input Vertices $u, u' \in V(T)$,

Output The deepest (furthest from the root) common ancestor of u, u' in T.

This query can be answered in $O(1)$ time by preprocessing in $O(n)$ space and $O(n)$ time using range minimum queries.

Tree Product Query Given a set S, a semigroup $\circ\colon S^2 \to S$, a tree T with n vertices, and a mapping $f\colon V(T) \to S$. The query defined by the following pair of input and output is called a Tree Product Query for $S, \circ, T, f$.

Input Vertices $u, u' \in V(T)$,

Output Let $u = v_1, v_2, \ldots, v_{k-1}, v_k = u'$ be the only path on T that connects u, u', then $f(v_1) \circ f(v_2) \circ \ldots \circ f(v_k)$.

This query can be answered in $O(\alpha(n))$ time by preprocessing in $O(n)$ space and $O(n)$ time [4]. Here, α is the inverse Ackermann function.

For a nonnegative integer i, ℓ, we define $A_\ell(i)$ as follows.

$$A_\ell(i) = \begin{cases} i+1 & \ell = 0, i \geq 0 \\ A_{\ell-1}^{(i+1)}(i+8) & \ell \geq 1, i \geq 0. \end{cases}$$

Note that $A_{\ell-1}^{(i+1)}$ denotes a function that $A_{\ell-1}$ iterated $i+1$ times. Using this, the inverse Ackermann function α is defined as $\alpha(n) = \min\{\ell \in \mathbb{Z}_{\geq 0} \mid A_\ell(1) > n\}$.

3 Triconnected component decomposition-based indexing

This section describes indices based on triconnected component decomposition for biconnected graphs. First, for each $\mu, \lambda \in \mathcal{T}$, we define $K_{\mu,\lambda} = \left(\{x_\mu, y_\mu\} \cup \{x_\lambda, y_\lambda\}, (\{x_\mu, y_\mu\} \cup \{x_\lambda, y_\lambda\})^2\right)$ to be a complete graph with self loops, and let $K_{\mu,\lambda}^{\vec{w}}$ denote the graph $K_{\mu,\lambda}$ with the weight

$$\vec{w} \colon V(K_{\mu,\lambda})^2 \to W^2 \ (\vec{w}(u,v) = \begin{pmatrix} w(u,v) \\ w^{\mathrm{B}}(u,v) \end{pmatrix}).$$

Also, let $\mathcal{K} = \left\{K_{\mu,\lambda}^{\vec{w}} \mid \mu, \lambda \in V(\mathcal{T}), \vec{w} \colon V(K_{\mu,\lambda})^2 \to W^2\right\} \cup \{\bot\}$ be the union of the set of those weighted graphs and $\{\bot\}$.

Next, for convenience, we define the maps $F_i \colon \mathrm{dom}(F_i) \to \mathcal{K}$ that provide data of distance and beer distance ($i = 1, 2, 3, 4$, specific domains are described later). The algorithms for beer path queries precompute some of these as data structures.

For each $\mathcal{X} \in \mathrm{dom}(F_i)$, let $F_i(\mathcal{X}) \in \mathcal{K}$ be a complete graph with at most 4 vertices and $\vec{f_i}(\mathcal{X})$ be its weight. Also, we will denote the weight $\vec{f_i}(\mathcal{X})(u,v)$ of each vertex pair $\langle u, v\rangle \in V(F_i(\mathcal{X}))^2$ by

$$\vec{f_i}(\mathcal{X}, u, v) = \begin{pmatrix} f_i(\mathcal{X}, u, v) \\ f_i^{\mathrm{B}}(\mathcal{X}, u, v) \end{pmatrix}$$

omitting some brackets. These maps are defined so that $f_i(\mathcal{X}, u, v)$ represents the normal distance and $f_i^{\mathrm{B}}(\mathcal{X}, u, v)$ represents the beer distance.

3.1 Definition of the mapping F_1 and its computation

▶ **Definition 2.** *We define the mapping $F_1 \colon V(\mathcal{T}) \setminus \{\rho\} \to \mathcal{K}$ as follows: For each node $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, let $F_1(\mu) = K_{\mu,\mu}^{\vec{f_1}(\mu)}$ (a complete graph consists of 2 vertices x_μ, y_μ). The weight of each vertex pair $\langle u, v\rangle \in \{x_\mu, y_\mu\}^2$ is $\vec{f_1}(\mu, u, v) = \vec{d}(G_\mu, u, v)$.*

The $F_1(\mu)$ intuitively represents the distance data when using the part of the $\mathcal{T}$ shown in Figure 2.We can compute F_1 from the leaves of $\mathcal{T}$ to the root as described below.

If $\mu \in Q_{\mathcal{T}} \setminus \{\rho\}$, G_μ is a graph that consists of only edges $\langle x_\mu, y_\mu\rangle, \langle y_\mu, x_\mu\rangle$, so we can calculate the weights as in

$$f_1(\mu, u, v) = w(u, v), \quad f_1^{\mathrm{B}}(\mu, u, v) = \begin{cases} \infty & B \cap \{x_\mu, y_\mu\} = \emptyset \\ \min_{p \in B \cap \{x_\mu, y_\mu\}} \{w(u,p) + w(p,v)\} & B \cap \{x_\mu, y_\mu\} \neq \emptyset. \end{cases}$$

From now on, we assume that μ is an inner node and that $F_1(\lambda)$ is computed for each of its child nodes $\lambda \in \mathrm{Ch}_\mu$. Also, let H_μ be the weighted graph with each edge $\langle x_\lambda, y_\lambda\rangle, \langle y_\lambda, x_\lambda\rangle$ ($\lambda \in \mathrm{Ch}_\mu$) of $\mathrm{Sk}_\mu \setminus \mathrm{Ref}_\mu$ given a weight $f_1(\mu, x_\mu, y_\mu)$, $f_1(\mu, y_\mu, x_\mu)$ respectively. Then, from

the definition of H_μ, if we consider the path from u to v in G_μ through the subgraph G_λ ($\langle u, v\rangle \in \{x_\lambda, y_\lambda\}^2$). The distance $f_1(\mu, u, v)$ can be obtained by referring to the weight of the edge $\langle u, v\rangle \in E(H_\mu)$ (the beer distance $f_1^{\mathrm{B}}(\mu, u, v)$ is obtained by referring to $f_1^{\mathrm{B}}(\lambda, u, v) = \mathrm{d}^{\mathrm{B}}(G_\lambda, u, v)$ directly). Therefore, $F_1(\mu)$ can be calculated by using H_μ instead of G_μ.

If $\mu \in S_{\mathcal{T}}$, let $\mathrm{Ch}_\mu = \{\mu_1, \ldots, \mu_k\}$ and let $x_\mu = x_{\mu_1}, y_{\mu_i} = x_{\mu_{i+1}}$ $(1 \leq i \leq k-1), y_{\mu_k} = y_\mu$ in Sk_μ (see Figure 7 of Appendix). Here, we define the following six symbols for each $\mu \in S_{\mathcal{T}}$:

- $\sigma_\mu^{\mathrm{xy}}[i, j] \coloneqq \begin{cases} \sum_{i \leq p \leq j} f_1(\mu_p, x_{\mu_p}, y_{\mu_p}) & 1 \leq i \leq j \leq k, \\ 0 & \text{otherwise}, \end{cases}$
  which is the distance from x_{μ_i} to y_{μ_j} in $\bigcup_{i \leq p \leq j} G_{\mu_p}$.
- $\sigma_\mu^{\mathrm{yx}}[i, j] \coloneqq \begin{cases} \sum_{i \leq p \leq j} f_1(\mu_p, y_{\mu_p}, x_{\mu_p}) & 1 \leq i \leq j \leq k, \\ 0 & \text{otherwise}, \end{cases}$
- $\beta_\mu^{\mathrm{xx}}[i] \coloneqq \sigma_\mu^{\mathrm{xy}}[1, i-1] + f_1^{\mathrm{B}}(\mu_i, x_{\mu_i}, x_{\mu_i}) + \sigma_\mu^{\mathrm{yx}}[1, i-1]$ $(1 \leq i \leq k$,
  which is the distance of the shortest walk that reaches from $x_\mu = x_{\mu_1}$ to $y_{\mu_{i-1}} = x_{\mu_i}$ in $\bigcup_{1 \leq j \leq i-1} G_{\mu_j}$, back to x_{μ_i} via a beer vertex in G_{μ_i} and again to x_μ.
- $\beta_\mu^{\mathrm{xy}}[i] \coloneqq f_1^{\mathrm{B}}(\mu_i, x_{\mu_i}, y_{\mu_i}) - f_1(\mu_i, x_{\mu_i}, y_{\mu_i})$ $(1 \leq i \leq k)$,
  which is the difference between the beer distance and the (mere) distance in moving from x_{μ_i} to y_{μ_i} in G_{μ_i}.
- $\beta_\mu^{\mathrm{yx}}[i] \coloneqq f_1^{\mathrm{B}}(\mu_j, y_{\mu_j}, x_{\mu_j}) - f_1(\mu_j, y_{\mu_j}, x_{\mu_j})$ $(1 \leq i \leq k)$,
- $\beta_\mu^{\mathrm{yy}}[i] \coloneqq \sigma_\mu^{\mathrm{yx}}[i+1, k] + f_1^{\mathrm{B}}(\mu_i, y_{\mu_i}, y_{\mu_i}) + \sigma_\mu^{\mathrm{xy}}[i+1, k]$ $(1 \leq i \leq k)$.

Note that we only preprocess $\sigma_\mu^{\mathrm{xy}}[1, i], \sigma_\mu^{\mathrm{xy}}[i, k], \sigma_\mu^{\mathrm{yx}}[1, i], \sigma_\mu^{\mathrm{yx}}[i, k]$ $(1 \leq i \leq k)$ among $\sigma_\mu^{\mathrm{xy}}[i, j]$, $\sigma_\mu^{\mathrm{yx}}[i, j]$. The other $\sigma_\mu^{\mathrm{xy}}[i, j]$ and $\sigma_\mu^{\mathrm{yx}}[i, j]$ are obtained and used in $O(1)$ time each time. We also preprocess $\beta_\mu^{\mathrm{xx}}[\cdot], \beta_\mu^{\mathrm{xy}}[\cdot], \beta_\mu^{\mathrm{yx}}[\cdot], \beta_\mu^{\mathrm{yy}}[\cdot]$ for Range Minimum Query. All of the above preprocessing can be computed in $O(k)$ space and $O(k)$ time. By using these, we can calculate the weights as in

$$\vec{f_1}(\mu, x_\mu, x_\mu) = \begin{pmatrix} 0 \\ \min_{1 \leq i \leq k} \{\beta_\mu^{\mathrm{xx}}[i]\} \end{pmatrix}, \quad \vec{f_1}(\mu, x_\mu, y_\mu) = \begin{pmatrix} \sigma_\mu^{\mathrm{xy}}[1, k] \\ \sigma_\mu^{\mathrm{xy}}[1, k] + \min_{1 \leq i \leq k} \{\beta_\mu^{\mathrm{xy}}[i]\} \end{pmatrix}.$$

If $\mu \in P_{\mathcal{T}}$, we define the following for each $\langle u, v\rangle \in \{x_\mu, y_\mu\}^2$ to simplify:

$$\ell_{u,v} = \min_{\lambda \in \mathrm{Ch}_\mu} \{f_1(\lambda, u, v)\}, \ \ell_{u,v}^{\mathrm{B}} = \min_{\lambda \in \mathrm{Ch}_\mu} \{f_1^{\mathrm{B}}(\lambda, u, v)\}.$$

By using these, we can calculate the weights as in

$$\vec{f_1}(\mu, x_\mu, x_\mu) = \begin{pmatrix} 0 \\ \min\{\ell_{x_\mu, x_\mu}^{\mathrm{B}}, \ \ell_{x_\mu, y_\mu} + \ell_{y_\mu, y_\mu}^{\mathrm{B}} + \ell_{y_\mu, x_\mu}, \ \ell_{x_\mu, y_\mu}^{\mathrm{B}} + \ell_{y_\mu, x_\mu}, \ \ell_{x_\mu, y_\mu} + \ell_{y_\mu, x_\mu}^{\mathrm{B}}\} \end{pmatrix},$$

$$\vec{f_1}(\mu, x_\mu, y_\mu) = \begin{pmatrix} \ell_{x_\mu, y_\mu} \\ \min\{\ell_{x_\mu, x_\mu}^{\mathrm{B}} + \ell_{x_\mu, y_\mu}, \ \ell_{x_\mu, y_\mu} + \ell_{y_\mu, y_\mu}^{\mathrm{B}}, \ \ell_{x_\mu, y_\mu}^{\mathrm{B}}, \ 2\ell_{x_\mu, y_\mu} + \ell_{y_\mu, x_\mu}^{\mathrm{B}}\} \end{pmatrix}.$$

If $\mu \in R_{\mathcal{T}}$, each weight $\vec{f_1}(\mu, u, v)$ can be calculated on H_μ as follows.

$$\vec{f_1}(\mu, u, v) = \begin{pmatrix} \mathrm{d}(H_\mu, u, v) \\ \min_{\lambda \in \mathrm{Ch}_\mu} \{\min_{p,q \in \{x_\lambda, y_\lambda\}} \{\mathrm{d}(H_\mu, u, p) + f_1^{\mathrm{B}}(\lambda, p, q) + \mathrm{d}(H_\mu, q, v)\}\} \end{pmatrix}.$$

Note that each $\mathrm{d}(H_\mu, a, b)$ in the above equation is calculated by a shortest path algorithm for H_μ. An example of calculating F_1 is shown in Figure 1.

3.2 Definition of the mapping F_2 and its computation

▶ **Definition 3.** *We define the mapping $F_2 \colon V(\mathcal{T}) \setminus \{\rho\} \to \mathcal{K}$ as follows: For each node $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, let $F_2(\mu) = K_\mu^{\vec{f_2}(\mu)}$ (a complete graph consists of 2 vertices x_μ, y_μ). The weight of each vertex pair $\langle u, v\rangle \in \{x_\mu, y_\mu\}^2$ is $\vec{f_2}(\mu, u, v) = \vec{d}(G \setminus E(G_\mu), u, v)$.*

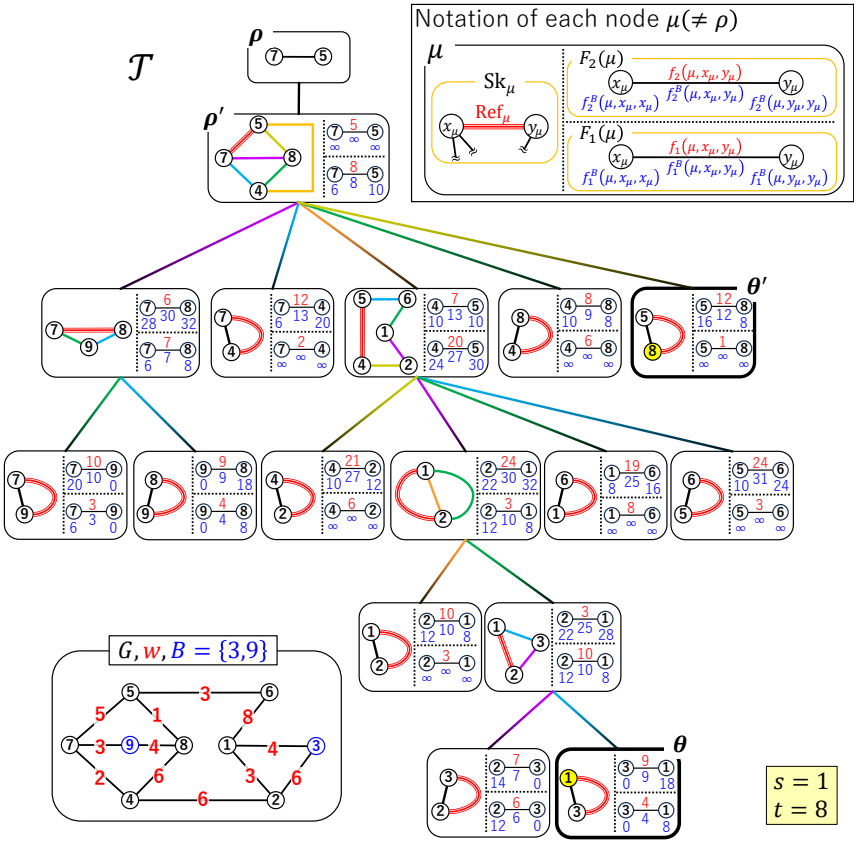


Figure 1 An weighted beer graph (G, w, B) (lower left) and its SPQR tree with F_1, F_2.

The $F_2(\mu)$ intuitively represents the distance data when using the part of the $\mathcal{T}$ shown in Figure 3. We can compute F_2 from the root of $\mathcal{T}$ to the leaves. To describe how to compute $F_2(\mu)$, let λ be the parent node of μ in $\mathcal{T}$.

If $\lambda = \rho$ (root node), the edges of $G \setminus E(G_\mu)$ are only $\langle x_\lambda, y_\lambda \rangle, \langle y_\lambda, x_\lambda \rangle$, so each weight $\vec{f_2}(\mu, u, v)$ can be calculated by replacing μ to λ in the formula for $\vec{f_1}(\mu, u, v)$.

From here, we assume that $\lambda \neq \rho$. Then, $F_2(\mu)$ can be calculated by using $H_\lambda \setminus \mathrm{Ref}_\mu \cup \mathrm{Ref}_\lambda$ instead of $G \setminus E(G_\mu)$. We set the weights of the edges $\langle x_\lambda, y_\lambda \rangle$ and $\langle y_\lambda, x_\lambda \rangle$ of this graph to $f_2(\lambda, x_\lambda, y_\lambda)$ and $f_2(\lambda, y_\lambda, x_\lambda)$, respectively.

For $\lambda \in S_\mathcal{T}$, let $\mathrm{Ch}_\lambda = \{\lambda_1, \ldots, \lambda_k\}, \mu = \lambda_i$, $x_\lambda = x_{\lambda_1}, y_{\lambda_j} = x_{\lambda_{j+1}}$ $(1 \leq j \leq k-1), y_{\lambda_k} = y_\lambda$ in Sk_λ. Each weights can be calculated in the same way as $\vec{f_1}$ for S nodes, for example,

$$f_2(\mu, x_{\lambda_i}, y_{\lambda_i}) = \sigma_\lambda^{\mathrm{yx}}[1, i-1] + f_2(\lambda, x_\lambda, y_\lambda) + \sigma_\lambda^{\mathrm{yx}}[i+1, k],$$

$$f_2^{\mathrm{B}}(\mu, x_{\lambda_i}, y_{\lambda_i}) = f_2(\mu, x_{\lambda_i}, y_{\lambda_i}) + \min \left\{ \begin{array}{c} \min\limits_{1 \leq j \leq k, j \neq i} \{\beta_\lambda^{\mathrm{yx}}[j]\} \\ f_2^{\mathrm{B}}(\lambda, x_\lambda, y_\lambda) - f_2(\lambda, x_\lambda, y_\lambda) \end{array} \right\}.$$

The weights F_2 for the P and R nodes can be calculated as well as F_1.

An example of calculating F_2 is shown in Figure 1.

3.3 Definition of the mapping F_3 and its computation

▶ **Definition 4.** *We define the mapping* $F_3 \colon E(\mathcal{T}) \setminus \{\{\rho, \rho'\}\} \to \mathcal{K}$ *as follows: For each edge* $\mathcal{E} = \{\mu, \lambda\} \in E(\mathcal{T}) \setminus \{\{\rho, \rho'\}\}$ *(*$\lambda \in \mathrm{Ch}_\mu$ *),* $F_3(\mathcal{E}) = K_{\mu,\lambda}^{\vec{f_3}(\mathcal{E})}$ *(a complete graph consists of at most 4 vertices). The weight of each vertex pair* $\langle u, v \rangle \in (\{x_\mu, y_\mu\} \cup \{x_\lambda, y_\lambda\})^2$ *is* $\vec{f_3}(\mathcal{E}, u, v) = \vec{d}(G_\mu \setminus E(G_\lambda), u, v)$.

The $F_3(\mathcal{E})$ intuitively represents the distance data when using the part of the $\mathcal{T}$ shown in Figure 4. In the actual $F_3(\mathcal{E})$ calculation, we can consider $H_\mu \setminus \mathrm{Ref}_\lambda$ instead of $G_\mu \setminus E(G_\lambda)$. $F4$ can be calculated by the same idea as the previous mappings. An example of calculating a part of F_3 is shown in Figure 8.

3.4 Definition of the mapping F_4 and its computation

▶ **Definition 5.** *We define the mapping* $F_4\colon \bigcup_{\mu\in V(\mathcal{T})\setminus Q_\mathcal{T}} \binom{\mathrm{Ch}_\mu}{2} \to \mathcal{K}$ *as follows: For each node* $\mu \in V(\mathcal{T}) \setminus Q_\mathcal{T}$ *and each node pair* $\psi = \{\lambda, \lambda'\} \in \binom{\mathrm{Ch}_\mu}{2}$ *of* μ, $F_4(\psi) = K^{\vec{f}_4(\psi)}_{\lambda,\lambda'}$ *(a complete graph consists of at most 4 vertices). The weight of each vertex pair* $\langle u, v\rangle \in (\{x_\lambda, y_\lambda\} \cup \{x_{\lambda'}, y_{\lambda'}\})^2$ *is* $\vec{f}_4(\psi, u, v) = \vec{d}(G \setminus E(G_\lambda) \setminus E(G_{\lambda'}), u, v)$.

The $F_4(\psi)$ intuitively represents the distance data when using the part of the $\mathcal{T}$ shown in Figure 5. In the actual $F_4(\psi)$ calculation, we can consider $H_\mu \setminus \mathrm{Ref}_\lambda \setminus \mathrm{Ref}_{\lambda'} \cup \mathrm{Ref}_\mu$ instead of $G \setminus E(G_\lambda) \setminus E(G_{\lambda'})$. We set the weights of the edges $\langle x_\mu, y_\mu\rangle, \langle y_\mu, x_\mu\rangle$ of this graph to $f_2(\mu, x_\mu, y_\mu)\,, f_2(\mu, y_\mu, x_\mu)$ respectively. F_4 can be calculated by the same idea as the previous mappings.

If $\mu \in S_\mathcal{T} \cup P_\mathcal{T}$, $F_4(\psi)$ can be computed in $O(1)$ time by using Range Minimum Query. Also, the beer distance is obtained from a graph that is a combination of the images of F_1, F_2, F_3, F_4, but the image of F_4 appears in at most one element of the combination (see subsection 4.2 for details). Therefore, it is enough to compute F_4 only for the child node pairs of the R node in the preprocessing. From this it is convenient to consider a mapping restricting the domain of F_4 and we define $F_{4\mathrm{R}}\colon \bigcup_{\mu\in R_\mathcal{T}} \binom{\mathrm{Ch}_\mu}{2} \to \mathcal{K}$ $(F_{4\mathrm{R}}(\psi) = F_4(\psi),$ $\psi \in \bigcup_{\mu\in R_\mathcal{T}} \binom{\mathrm{Ch}_\mu}{2})$.

Because of space limitation, we show analyses on computational complexities in Section A.3.

4 Algorithm based on triconnected component decomposition

4.1 Definition of binary operations

We define the binary operation $\oplus\colon \mathcal{K}^2 \to \mathcal{K}$ as follows.

▶ **Definition 6.** *For each* $H_1, H_2 \in \mathcal{K}$, $H_1 \oplus H_2$ *is defined as follows. If* $H_1 = \perp$ *or* $H_2 = \perp$, $H_1 \oplus H_2 = \perp$. *If* $H_1 \neq \perp$ *and* $H_2 \neq \perp$, *let* $H_i = K^{\vec{w}_i}_{\mu_i,\lambda_i}$ *(*$i = 1, 2$*). Also, let* $A = (\{\mu_1\} \cup \{\lambda_1\}) \cap (\{\mu_2\} \cup \{\lambda_2\})$ *be the set of nodes in* $\mathcal{K}$ *that give vertices appearing in* H_1, H_2 *in common. If* $|A| \neq 1$, $H_1 \oplus H_2 = \perp$. *If* $|A| = 1$, *let* $A = \{\theta\}$, $H_1 \oplus H_2 = K^{\vec{w}}_{\theta_1,\theta_2}$. *Here, we define* θ_1, θ_2 *as follows.*

$$\theta_1 = \begin{cases} \mu_1 (= \lambda_1) & \mu_1 = \theta = \lambda_1 \\ \mu_1 & \mu_1 \neq \theta = \lambda_1 \\ \lambda_1 & \mu_1 = \theta \neq \lambda_1 \end{cases}, \quad \theta_2 = \begin{cases} \mu_2 (= \lambda_2) & \mu_2 = \theta = \lambda_2 \\ \mu_2 & \mu_2 \neq \theta = \lambda_2 \\ \lambda_2 & \mu_2 = \theta \neq \lambda_2 \end{cases}.$$

Also, let $H_1 \tilde{\cup} H_2$ *be a weighted multi graph with vertex set* $V(H_1) \cup V(H_2)$, *given distinct edges in* H_1 *and edges in* H_2, *and let* $\vec{z}$ *be the weights defined by*

$$\vec{z}(e) = \begin{pmatrix} z(e) \\ z^{\mathrm{B}}(e) \end{pmatrix} = \vec{w}_i(p, q) \ (e = \langle p, q\rangle \in E(H_i), i = 1, 2).$$

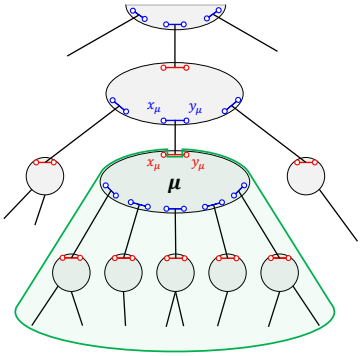


Figure 2 The subtree considered in $F_1(\mu)$.

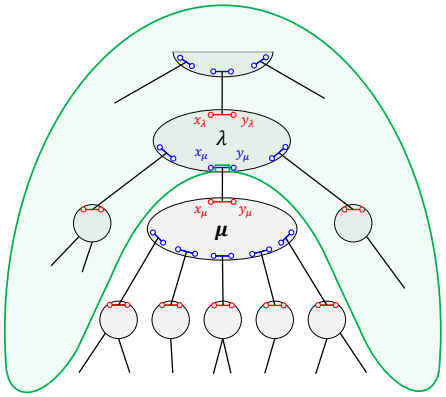


Figure 3 The subtree considered in $F_2(\mu)$.

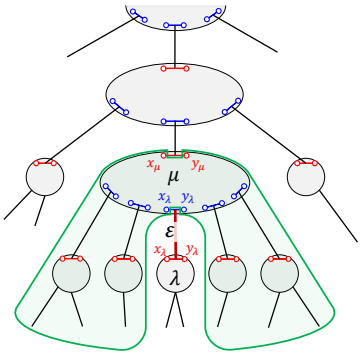


Figure 4 The subtree considered in $F_3(\mathcal{E})$.

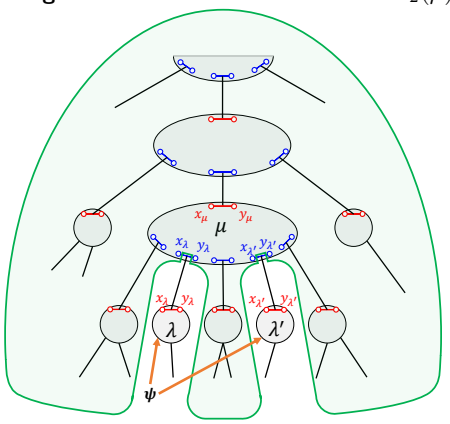


Figure 5 The subtree considered in $F_4(\psi)$.

Then, For each $\langle u, v\rangle \in (\{x_{\theta_1}, y_{\theta_1}\} \cup \{x_{\theta_2}, y_{\theta_2}\})^2$*, we define* $\vec{w}(u, v)$ *as follows.*

$$w(u,v) = \mathrm{d}((H_1 \tilde{\cup} H_2, z), u, v)\,,$$
$$w^{\mathrm{B}}(u,v) = \min_{e=\langle p,q\rangle \in E(H_1 \tilde{\cup} H_2)} \{\mathrm{d}((H_1 \tilde{\cup} H_2, z), u, p) + z^{\mathrm{B}}(e) + \mathrm{d}((H_1 \tilde{\cup} H_2, z), q, v)\}.$$

For a concrete example of this operation, see the computation of $H \oplus H'$ in Figure 8. Furthermore, we define a subset $\widehat{\mathcal{K}}$ of $\mathcal{K}$ by $\widehat{\mathcal{K}} = \{K^{\vec{w}}_{\mu,\lambda} \in \mathcal{K} \setminus \{\bot\} \mid \lambda \in \mathrm{Des}_\mu\} \cup \{\bot\}$ and define the binary operation $\widehat{\oplus}\colon \widehat{\mathcal{K}}^2 \to \widehat{\mathcal{K}}$ as follows.

▶ **Definition 7.** *For each* $H_1, H_2 \in \widehat{\mathcal{K}}$*, if* $H_1 = \bot$ *or* $H_2 = \bot$*, then* $H_1 \widehat{\oplus} H_2 = \bot$*, otherwise, let* $H_i = K^{\vec{w}_i}_{\mu_i,\lambda_i}, \lambda_i \in \mathrm{Des}_{\mu_i}$ *(*$i = 1, 2$*) then*

$$H_1 \widehat{\oplus} H_2 = \begin{cases} \bot & \lambda_1 \neq \mu_2 \\ H_1 \oplus H_2 & \lambda_1 = \mu_2. \end{cases}$$

▶ **Lemma 8.** $\widehat{\oplus}$ *is a semigroup.*

The proof is given in Section A.1.

4.2 Representation of distance and beer distance using mapping and algorithms for Beer Path Query

By using $F_1, F_2, F_3, F_4, F_{4\mathrm{R}}$ and tree product query data structures, we can compute beer path between given vertices. Recall that r is the maximum edge number of the skeleton of R nodes in the SPQR tree of G and W is the range of the edge weight function.

▶ **Theorem 9.** *If we precompute F_1, F_2 as a data structure, the space required to store the data structure is $O(m)$. The preprocessing time and query time are*
1. *$O(m+r\cdot\min\{m,rn\})$ and $O(n+r\cdot\min\{m,rn\}+\alpha(m))$ if $W=\mathbb{Z}_{\geq 0}$ and G is undirected.*
2. *$O(m+r(m+n\log r_+))$ and $O(n+r(m+n\log r_+)+\alpha(m))$ if $W=\mathbb{R}_{\geq 0}$.*

▶ **Theorem 10.** *If we precompute F_1, F_2, F_3 as a data structure, the space required to store the data structure is $O(m)$. The preprocessing time and query time are*
1. *$O(m+r^2\cdot\min\{m,rn\})$ and $O(r^2+\alpha(m))$ if $W=\mathbb{Z}_{\geq 0}$ and G is undirected.*
2. *$O(m+r^2(m+n\log r_+))$ and $O(r^2\log r_+ +\alpha(m))$ if $W=\mathbb{R}_{\geq 0}$.*

▶ **Theorem 11.** *If we precompute $F_1, F_2, F_3, F_{4\mathrm{R}}$ as a data structure, the space required to store the data structure is $O(m+r\cdot\min\{m,rn\})$. The preprocessing time and query time are*
1. *$O(m+r^3\cdot\min\{m,rn\})$ and $O(\alpha(m))$ if $W=\mathbb{Z}_{\geq 0}$ and G is undirected.*
2. *$O(m+r^3(m+n\log r_+))$ and $O(\alpha(m))$ if $W=\mathbb{R}_{\geq 0}$.*

Proofs are given in Section A.4

References

1 Joyce Bacic, Saeed Mehrabi, and Michiel Smid. Shortest beer path queries in outerplanar graphs. *Algorithmica*, 85(6):1679–1705, 2023.

2 Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM Journal on Computing*, 22(2):221–242, 1993.

3 Hans L Bodlaender. Treewidth: Algorithmic techniques and results. In *Mathematical Foundations of Computer Science 1997: 22nd International Symposium, MFCS'97 Bratislava, Slovakia, August 25–29, 1997 Proceedings 22*, pages 19–36. Springer, 1997.

4 Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2(1-4):337–361, 1987.

5 Rathish Das, Meng He, Eitan Kondratovsky, J. Ian Munro, Anurag Murty Naredla, and Kaiyu Wu. Shortest Beer Path Queries in Interval Graphs. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 59:1–59:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

6 Arash Farzan and Shahin Kamali. Compact navigation and distance oracles for graphs with small treewidth. *Algorithmica*, 69(1):92–116, 2014.

7 Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

8 Carsten Gutwenger and Petra Mutzel. A linear time implementation of SPQR-trees. In Joe Marks, editor, *Graph Drawing*, pages 77–90, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

9 J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.

10 Manas Jyoti Kashyop, Tsunehiko Nagayama, and Kunihiko Sadakane. Faster algorithms for shortest path and network flow based on graph decomposition. *J. Graph Algorithms Appl.*, 23(5):781–813, 2019.

11 Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM (JACM)*, 46(3):362–394, 1999.

A Missing Proofs

A.1 Proof of Lemma 8

Proof. We arbitrarily take $H_1, H_2, H_3 \in \widehat{\mathcal{K}}$, and confirm that $H_{(12)3} \coloneqq (H_1 \widehat{\oplus} H_2) \widehat{\oplus} H_3$ and $H_{1(23)} \coloneqq H_1 \widehat{\oplus} (H_2 \widehat{\oplus} H_3)$ are equal. First, if $H_1 = \bot$ or $H_2 = \bot$ or $H_3 = \bot$, clearly $H_{(12)3} = H_{1(23)} = \bot$. In the following, let $H_i \neq \bot$ and $H_i = K^{\vec{w}_i}_{\mu_i,\lambda_i}$ ($\lambda_i \in \mathrm{Des}_{\mu_i}$) for each i.

If $\lambda_1 \neq \mu_2$ or $\lambda_2 \neq \mu_3$, then we can easily obtain $H_{(12)3} = H_{1(23)} = \perp$. If $\lambda_1 = \mu_2, \lambda_2 = \mu_3$, let $H_{(12)3} = K^{\vec{w}_{(12)3}}_{\mu_1,\lambda_3}, H_{1(23)} = K^{\vec{w}_{1(23)}}_{\mu_1,\lambda_3}$. Then, we show briefly that $\vec{w}_{(12)3}(u,v) = \vec{w}_{1(23)}(u,v)$ for each $u, v \in \{x_{\mu_1}, y_{\mu_1}\} \cup \{x_{\lambda_3}, y_{\lambda_3}\}$.

If let $H_1 \widehat{\oplus} H_2 = H_{12} = K^{\vec{w}_{12}}_{\mu_1,\mu_3}$, then from Figure 9 the following holds for each $u' \in \{x_{\mu_1}, y_{\mu_1}\}$ and $v' \in \{x_{\mu_3}, y_{\mu_3}\}$.

$$w_{12}(u',v') = \mathrm{d}(H_{12},u',v') = \min_{p\in\{x_{\mu_2},y_{\mu_2}\}} \{\mathrm{d}(H_1,u',p) + \mathrm{d}(H_2,p,v')\}.$$

By using this, the following holds for each $u \in \{x_{\mu_1}, y_{\mu_1}\}$ and $v \in \{x_{\lambda_3}, y_{\lambda_3}\}$.

$$\begin{aligned} w_{(12)3}(u,v) = \mathrm{d}\big(H_{(12)3},u,v\big) &= \min_{q\in\{x_{\mu_3},y_{\mu_3}\}} \{\mathrm{d}(H_{12},u,q) + \mathrm{d}(H_3,q,v)\} \\ &= \min_{q\in\{x_{\mu_3},y_{\mu_3}\}} \left\{ \min_{p\in\{x_{\mu_2},y_{\mu_2}\}} \{\mathrm{d}(H_1,u,p) + \mathrm{d}(H_2,p,q)\} + \mathrm{d}(H_3,q,v) \right\} \\ &= \min_{p\in\{x_{\mu_2},y_{\mu_2}\},q\in\{x_{\mu_3},y_{\mu_3}\}} \{\mathrm{d}(H_1,u,p) + \mathrm{d}(H_2,p,q) + \mathrm{d}(H_3,q,v)\}. \end{aligned}$$

By the same idea, exactly the same result is obtained for $w_{1(23)}(u,v)$. We can show $w_{(12)3}(u,v) = w_{1(23)}(u,v)$ and $w^{\mathrm{B}}_{(12)3}(u,v) = w^{\mathrm{B}}_{1(23)}(u,v)$ for the other weights in the same way. ◀

A.2 Preprocessing algorithms

A.2.1 Algorithm for preprocessing F_1, F_2

We consider an algorithm that preprocesses F_1, F_2. For this algorithm, the preprocessing space is $C_1^{\mathrm{space}} + C_2^{\mathrm{space}} = O(m)$ and the preprocessing time is

$$C_1^{\mathrm{time}} + C_2^{\mathrm{time}} = \begin{cases} O(m + r \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0} \\ O(m + r(m + n\log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases}$$

Beer Path Query can be solved by computing $O(n)$ images of F_3 and one image of F_4 on Π and combine them using Tree Product Query. Thus, query time is

$$\begin{aligned} &O\left(\textstyle\sum_{\mu\in S_{\mathcal{T}}\cup P_{\mathcal{T}}} 1 + \sum_{\mu\in R_{\mathcal{T}}} m_\mu \mathrm{ALG}\left(H_\mu\right) + m_\pi \mathrm{ALG}\left(H_\pi\right) + \alpha(m)\right) \\ &= \begin{cases} O(n + r \cdot \min\{m, rn\} + \alpha(m)) & W = \mathbb{Z}_{\geq 0} \\ O(n + r(m + n\log r_+) + \alpha(m)) & W = \mathbb{R}_{\geq 0}. \end{cases} \end{aligned}$$

A.2.2 Algorithm for preprocessing F_1, F_2, F_3

We consider an algorithm that preprocesses F_1, F_2, F_3. For this algorithm, the preprocessing space is $C_1^{\mathrm{space}} + C_2^{\mathrm{space}} + C_3^{\mathrm{space}} = O(m)$ and the preprocessing time is

$$C_1^{\mathrm{time}} + C_2^{\mathrm{time}} + C_3^{\mathrm{time}} = \begin{cases} O(m + r^2 \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0} \\ O(m + r^2(m + n\log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases}$$

Beer Path Query can be solved by computing one image of F_4 and combine images of F_3 on Π and F_4 using Tree Product Query. Thus, query time is

$$O(m_\pi \mathrm{ALG}\left(H_\pi\right) + \alpha(m)) = \begin{cases} O(r^2 + \alpha(m)) & W = \mathbb{Z}_{\geq 0} \\ O(r^2 \log r_+ + \alpha(m)) & W = \mathbb{R}_{\geq 0}. \end{cases}$$

A.2.3 Algorithm for preprocessing F_1, F_2, F_3, F_{4R}

We consider an algorithm that preprocesses F_1, F_2, F_3, F_{4R}. For this algorithm, the preprocessing space is $C_1^{\text{space}} + C_2^{\text{space}} + C_3^{\text{space}} + C_{4R}^{\text{space}} = O(m + r \cdot \min\{m, rn\})$ and the preprocessing time is

$$C_1^{\text{time}} + C_2^{\text{time}} + C_3^{\text{time}} + C_{4R}^{\text{time}} = \begin{cases} O(m + r^3 \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0} \\ O(m + r^3(m + n \log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases}$$

Beer Path Query can be solved by combining images of F_3 and F_4 on Π using Tree Product Query. Thus, query time is $O(\alpha(m))$.

A.3 Computational complexity for each mapping

In this subsection, we analyze the computational complexity for each mapping. Let C_i^{time} and C_i^{space} be the time required to compute each F_i and the space to store, respectively.

A.3.1 Computational complexity for F_1

First, we consider C_1^{space}. For each $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, $F_1(\mu)$ is a graph of constant size. Also, each $\mu \in S_\mathcal{T} \cup P_\mathcal{T}$ uses $O(m_\mu)$ space in the preprocessing for Range Minimum Query and so on. Thus, $C_1^{\text{space}} = O\left(\sum_{\mu \in Q_\mathcal{T} \cup R_\mathcal{T}} 1 + \sum_{\mu \in S_\mathcal{T} \cup P_\mathcal{T}} m_\mu\right) = O(m)$. Next, we consider C_1^{time}. If $\mu \in Q_\mathcal{T} \setminus \{\rho\}$, $F_1(\mu)$ can be computed in $O(1)$ time. If $\mu \in S_\mathcal{T} \cup P_\mathcal{T}$, preprocessing and $F_1(\mu)$ calculation can be done in $O(m_\mu)$ time. Also, if $\mu \in R_\mathcal{T}$, $F_1(\mu)$ can be obtained by running the shortest path algorithm for H_μ for $O(m_\mu)$ times, so it can be computed in $O\left(m_\mu \text{ALG}\left(H_\mu\right)\right)$ time. Thus, noting the definition of r, C_1^{time} is as follows:

$$\begin{aligned} C_1^{\text{time}} &= O\left(\sum_{\mu \in Q_\mathcal{T}} 1 + \sum_{\mu \in S_\mathcal{T} \cup P_\mathcal{T}} m_\mu + \sum_{\mu \in R_\mathcal{T}} m_\mu \text{ALG}\left(H_\mu\right)\right) \\ &= O\left(m + m + r \sum_{\mu \in R_\mathcal{T}} \text{ALG}\left(H_\mu\right)\right) = \begin{cases} O(m + r \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0} \\ O(m + r(m + n \log r_+)) & W = \mathbb{R}_{\geq 0} \end{cases}. \end{aligned}$$

A.3.2 Computational complexity for F_2

First, C_2^{space} can be similarly considered to C_1^{space}, $C_2^{\text{space}} = \sum_{\mu \in V(\mathcal{T}) \setminus \{\rho\}} O(1) = O(m)$. Next, we consider C_2^{time}. Let λ be the parent node of μ in $\mathcal{T}$. If $\lambda \in \{\rho\} \cup S_\mathcal{T} \cup P_\mathcal{T}$, $F_2(\mu)$ can be computed in $O(1)$ time by using Range Minimum Query. Also, if $\lambda \in R_\mathcal{T}$, $F_2(\mu)$ can be obtained by running the shortest path algorithm for $H_\lambda \setminus \text{Ref}_\mu \cup \text{Ref}_\lambda$ for $O(m_\lambda)$ times, so it can be computed in $O\left(m_\lambda \text{ALG}\left(H_\lambda\right)\right)$ time. Thus, C_2^{time} can be evaluated as follows:

$$\begin{aligned} C_2^{\text{time}} &= O\left(\sum_{\lambda \in \{\rho\} \cup S_\mathcal{T} \cup P_\mathcal{T}} 1 + \sum_{\lambda \in R_\mathcal{T}} m_\lambda \text{ALG}\left(H_\lambda\right)\right) = O\left(n + r \sum_{\lambda \in R_\mathcal{T}} \text{ALG}\left(H_\lambda\right)\right) \\ &= \begin{cases} O(n + r \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0}, \\ O(n + r(m + n \log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases} \end{aligned}$$

A.3.3 Computational complexity for F_3

First, we consider C_3^{space}. In F_3, a graph of a constant size is prepared for each edge of $E(\mathcal{T}) \setminus \{\{\rho, \rho'\}\}$, so $C_3^{\text{space}} = O(|E(\mathcal{T})|) = O(m)$. Next, we consider C_3^{time}. For each $\mathcal{E} = \{\mu, \lambda\} \in E(\mathcal{T}) \setminus \{\{\rho, \rho'\}\}$ ($\lambda \in \text{Ch}_\mu$), if $\mu \in S_\mathcal{T} \cup P_\mathcal{T}$ then $F_3(\mathcal{E})$ can be computed in $O(1)$ time by using Range Minimum Query. If $\mu \in R_\mathcal{T}$ then $F_3(\mathcal{E})$ can be obtained by running the shortest path algorithm for $H_\mu \setminus \text{Ref}_\lambda$ for $O(m_\lambda)$ times, so it can be computed in $O\left(m_\lambda \text{ALG}\left(H_\lambda\right)\right)$ time. Thus, C_3^{time} can be evaluated as follows.

$$C_3^{\text{time}} = O\left(\sum_{\mu\in S_{\mathcal{T}}\cup P_{\mathcal{T}}}\sum_{\lambda\in \mathrm{Ch}_\mu} 1 + \sum_{\mu\in R_{\mathcal{T}}}\sum_{\lambda\in \mathrm{Ch}_\mu} m_\mu \mathrm{ALG}\left(H_\mu\right)\right)$$
$$= O\left(m + r^2\sum_{\mu\in R_{\mathcal{T}}}\mathrm{ALG}\left(H_\mu\right)\right) = \begin{cases} O(m + r^2\cdot\min\{m, rn\}) & W = \mathbb{Z}_{\geq 0}, \\ O(m + r^2(m + n\log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases}$$

A.3.4 Computational complexity for F_4

If F_4 is realized as a data structure, it is not efficient because it requires computation and space even for images that can be computed in $O(1)$ time, as described in Subsection 3.4. Therefore, we consider the computational complexity of realizing $F_{4\mathrm{R}}$ as a data structure instead of F_4 itself.

First, the space for $F_{4\mathrm{R}}$ is $C_{4\mathrm{R}}^{\text{space}} = \sum_{\mu\in R_{\mathcal{T}}} O\left(m_\mu^2\right) = O(r\cdot\min\{m, rn\})$. Next, we consider the preprocessing time of $F_{4\mathrm{R}}$, $C_{4\mathrm{R}}^{\text{space}}$. For each $\mu\in R_{\mathcal{T}}$ and each node pair $\psi = \{\lambda, \lambda'\} \in \binom{\mathrm{Ch}_\mu}{2}$, $F_{4\mathrm{R}}(\psi) = F_4(\psi)$ can be obtained by running the shortest path algorithm for $H_\mu \setminus \mathrm{Ref}_\lambda \setminus \mathrm{Ref}_{\lambda'} \cup \mathrm{Ref}_\mu$ for $O(m_\mu)$ times, so it can be computed in $O\left(m_\mu \mathrm{ALG}\left(H_\mu\right)\right)$ time. Thus, $C_{4\mathrm{R}}^{\text{time}}$ is as follows:

$$C_{4\mathrm{R}}^{\text{time}} = O\left(\sum_{\mu\in R_{\mathcal{T}}}\sum_{\psi\in\binom{\mathrm{Ch}_\mu}{2}} m_\mu \mathrm{ALG}\left(H_\mu\right)\right) = O\left(\sum_{\mu\in R_{\mathcal{T}}} m_\mu^3 \mathrm{ALG}\left(H_\mu\right)\right)$$
$$= O\left(r^3\sum_{\mu\in R_{\mathcal{T}}}\mathrm{ALG}\left(H_\mu\right)\right) = \begin{cases} O(r^3\cdot\min\{m, rn\}) & W = \mathbb{Z}_{\geq 0}, \\ O(r^3(m + n\log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases}$$

A.4 Proofs for the algorithms

In the following, we describe an outline of the algorithm corresponding to each of the above theorems.

First, we describe the representation of distances and beer distances using each mapping F_i and binary operations. We also show several algorithms for Beer Path Query based on them. We consider computing the distance or beer distance from s to t in a biconnected connected graph G. First, let $\theta\in Q_{\mathcal{T}}\setminus\{\rho\}$ be a Q node whose skeleton contains vertex s, and let $\{x_\theta, y_\theta\} = \{s, s'\}$. Similarly, take a node $\theta'\in Q_{\mathcal{T}}\setminus\{\rho\}$ whose skeleton contains a vertex t and let $\{x_{\theta'}, y_{\theta'}\} = \{t, t'\}$.

If $\theta = \theta'$, then we combine $F_1(\theta)$ which contains data on the distance and the beer distance in G_θ, and $F_2(\theta)$ which contains data in $G\setminus E(G_\theta)$. The combined result is represented by $F_1(\theta)\oplus F_2(\theta)$, and the distance and beer distance are obtained by referring to the weights of the vertex pair $\langle s, t\rangle$ in $F_1(\theta)\oplus F_2(\theta)$.

If $\theta\neq\theta'$, let π be the lowest common ancestor of θ and θ' in $\mathcal{T}$ and denote the θ-θ' path in $\mathcal{T}$ by the vertex sequence Π: $\theta = \mu_k, \mu_{k-1}, \ldots, \mu_2, \mu_1, \pi, \lambda_1, \lambda_2, \ldots, \lambda_{\ell-1}, \lambda_\ell = \theta'$. $F_1(\theta) = F_1(\mu_k)$ contains the data of the distance and the beer distance of each pair of $\{s, s'\} = \{x_{\mu_k}, y_{\mu_k}\}$ in G_{μ_k}. Also, $F_3(\{\mu_{k-1}, \mu_k\})$ contains the data of each pair of $\{x_{\mu_{k-1}}, y_{\mu_{k-1}}\}\cup\{x_{\mu_k}, y_{\mu_k}\}$ in $G_{\mu_{k-1}}\setminus E(G_{\mu_k})$. Therefore, by combining $F_1(\mu_k)$ and $F_3(\{\mu_{k-1}, \mu_k\})$, we can obtain the data of each pair of $\{x_{\mu_{k-1}}, y_{\mu_{k-1}}\}\cup\{s, s'\}$ in $G_{\mu_{k-1}}$. And the combined result can be expressed as $F_1(\mu_k)\oplus F_3(\{\mu_{k-1}, \mu_k\})$.

By applying this idea repeatedly, we can obtain the data of each pair of $\{x_{\mu_1}, y_{\mu_1}\}\cup\{s, s'\}$ in G_{μ_1} by computing

$$F_1(\mu_k)\oplus\left(F_3(\{\mu_1, \mu_2\})\oplus\cdots\oplus F_3(\{\mu_{k-1}, \mu_k\})\right) = F_1(\mu_k)\oplus\left(\oplus_{i=1}^{k-1} F_3(\{\mu_i, \mu_{i+1}\})\right).$$

Similarly, by computing

$$(F_3(\{\lambda_1,\lambda_2\}) \oplus \cdots \oplus F_3(\{\lambda_{\ell-1},\lambda_\ell\})) \oplus F_1(\lambda_\ell) = \left(\oplus_{j=1}^{\ell-1} F_3(\{\lambda_j,\lambda_{j+1}\})\right) \oplus F_1(\lambda_\ell),$$

we can obtain the data of each pair of $\{x_{\lambda_1}, y_{\lambda_1}\} \cup \{t, t'\}$ in G_{λ_1}.

Furthermore, $F_4(\{\mu_1,\lambda_1\})$ contains the data of each pair of $\{x_{\mu_1}, y_{\mu_1}\} \cup \{x_{\lambda_1}, y_{\lambda_1}\}$ in $G \backslash E(G_{\mu_1}) \backslash E(G_{\lambda_1})$. Therefore, by combining this and the results of the above two operations, we can obtain the data of each pair of $\{s, s'\} \cup \{t, t'\}$ in G. And the combined result can be expressed as

$$K_{\theta,\theta'}^{\vec{w}_{s,t}} = F_1(\mu_k) \oplus \left(\oplus_{i=1}^{k-1} F_3(\{\mu_i,\mu_{i+1}\})\right) \oplus F_4(\{\mu_1,\lambda_1\}) \oplus \left(\oplus_{j=1}^{\ell-1} F_3(\{\lambda_j,\lambda_{j+1}\})\right) \oplus F_1(\lambda_\ell).$$

Then, we obtain the distance and the beer distance by $\mathrm{d}(G,s,t) = w_{s,t}(s,t), \mathrm{d}^{\mathrm{B}}(G,s,t) = w_{s,t}^{\mathrm{B}}(s,t)$.

Here, the computation of $K_{\theta,\theta'}^{\vec{w}_{s,t}}$ can be written

$$K_{\theta,\theta'}^{\vec{w}_{s,t}} = F_1(\mu_k) \oplus \left(\widehat{\oplus}_{i=1}^{k-1} F_3(\{\mu_i,\mu_{i+1}\})\right) \oplus F_4(\{\mu_1,\lambda_1\}) \oplus \left(\widehat{\oplus}_{j=1}^{\ell-1} F_3(\{\lambda_j,\lambda_{j+1}\})\right) \oplus F_1(\lambda_\ell)$$

by using the semigroup $\widehat{\oplus}$. Therefore, if we preprocess $\mathcal{T}$ for Tree Product Query regarding $\widehat{\oplus}$, we can compute $K_{\theta,\theta'}^{\vec{w}_{s,t}}$ for the $\oplus, \widehat{\oplus}$ operation in $O(\alpha(|V(\mathcal{T})|)) = O(\alpha(m))$ time.

B Algorithm based on tree decomposition

In this section, we describe algorithms based on tree decomposition. For a graph G with n vertices and m edges, denote its treewidth by $t \coloneqq \mathrm{tw}(G)$. Also, let $\mathcal{T}$ be a rooted tree decomposition of G with width t and $O(tn)$ nodes. Then, the following theorem holds.

▶ **Theorem 12.**

1. *When we construct a data structure in $O(t^3 n)$ space with preprocessing using $O(t^8 n)$ time, we can answer a query in $O(t^8 n + \alpha(tn))$ time.*
2. *When we construct a data structure in $O(t^3 n)$ space with preprocessing using $O(t^8 n)$ time, we can answer a query in $O(t^7 + \alpha(tn))$ time.*
3. *When we construct a data structure in $O(t^5 n)$ space with preprocessing using $O(t^{10} n)$ time, we can answer a query in $O(t^6 + \alpha(tn))$ time.*

In the following, we describe an outline of the proof of the above theorems. For each node μ in $\mathcal{T}$, let X_μ be the vertex subset of G that μ has, and let $S_\mu = X_\mu \cup \bigcup_{\lambda \in \mathrm{Des}_\mu} X_\lambda$. Furthermore, let A_μ be a vertex in X_μ if μ is a root node, and if μ is not the root node, $A_\mu = X_\mu \cap X_\lambda$ where $\mu \in \mathrm{Ch}_\lambda$ (A_μ corresponds to the endpoint set of Ref_μ in the SPQR tree). Then, we define the following symbols as well as the mapping to the SPQR tree:

$$\begin{aligned}
\vec{f_1}(\mu,u,v) &\coloneqq \vec{d}(G[S_\mu],u,v)\ (\mu \in V(\mathcal{T}), u,v \in A_\mu),\\
\vec{f_2}(\mu,u,v) &\coloneqq \vec{d}(G \setminus E(G[S_\mu]),u,v)\ (\mu \in V(\mathcal{T}), u,v \in A_\mu),\\
\vec{f_3}(\{\mu,\lambda\},u,v) &\coloneqq \vec{d}(G[S_\mu] \setminus E(G[S_\lambda]),u,v)\ ((\mu,\lambda) \in E(\mathcal{T}), \lambda \in \mathrm{Ch}_\mu, u,v \in A_\mu \cup A_\lambda),\\
\vec{f_4}(\{\lambda,\lambda'\},u,v) &\coloneqq \vec{d}(G \setminus E(G[S_\lambda]) \setminus E(G[S_{\lambda'}]),u,v)\\
&\qquad (\{\lambda,\lambda'\} \in \tbinom{\mathrm{Ch}_\mu}{2}, \mu \in V(\mathcal{T}), u,v \in A_\lambda \cup A'_\lambda).
\end{aligned}$$

We can calculate $\vec{f_1}$ as follows: If μ is a leaf of $\mathcal{T}$, then $f_1(\mu,u,v) = \mathrm{d}(G[X_\mu],u,v)$ and

$$f_1^{\mathrm{B}}(\mu,u,v) = \begin{cases} \min_{p \in B \cap X_\mu} \{\mathrm{d}(G[X_\mu],u,p) + \mathrm{d}(G[X_\mu],p,v)\} & B \cap X_\mu \neq \emptyset \\ \infty & B \cap X_\mu = \emptyset. \end{cases}$$

If μ is not leaves, then

$$f_1(\mu,u,v) = \min\left\{\begin{array}{c}\mathrm{d}(G[X_\mu],u,v)\\ \min_{\substack{\lambda\in \mathrm{Ch}_\mu\\ p,q\in A_\lambda}}\{\mathrm{d}(G[X_\mu],u,p)+f_1(\lambda,p,q)+\mathrm{d}(G[X_\mu],q,v)\}\end{array}\right\},$$

$$f_1^{\mathrm{B}}(\mu,u,v) = \begin{cases}\min\left\{\ell_{u,v}, \min_{p\in B\cap X_\mu}\{\mathrm{d}(G[X_\mu],u,p)+\mathrm{d}(G[X_\mu],p,v)\}\right\} & B\cap X_\mu\neq\emptyset\\ \ell_{u,v} & B\cap X_\mu=\emptyset,\end{cases}$$

where $\ell_{u,v} = \min_{\lambda\in\mathrm{Ch}_\mu,p,q\in A_\lambda}\{\mathrm{d}(G[X_\mu],u,p)+f_1^{\mathrm{B}}(\lambda,p,q)+\mathrm{d}(G[X_\mu],q,v)\}$. $\vec{f}_2,\vec{f}_3,\vec{f}_4$ can be calculated using the same idea.

For each $\mu\in V(\mathcal{T})$, $|X_\mu| = O(t), |E(G[X_\mu])| = O(t^2)$, so $\mathrm{ALG}\,(G[X_\mu]) = O(t^2)$ is obtained regardless of how we take the range W of the weights. Noting this, the space C_i^{space} and computation time C_i^{time} required for each $\vec{f}_i$ can be evaluated as follows:

$$C_1^{\mathrm{space}}, C_2^{\mathrm{space}} = \sum_{\mu\in V(\mathcal{T})} O\left(|A_\mu|^2\right) = O(t^2|V(\mathcal{T})|) = O(t^3 n),$$

$$C_3^{\mathrm{space}} = \sum_{\mathcal{E}\in E(\mathcal{T})} O(t^2) = O(t^2|E(\mathcal{T})|) = O(t^3 n),$$

$$C_4^{\mathrm{space}} = \sum_{\mu\in V(\mathcal{T})}\sum_{\psi\in\binom{\mathrm{Ch}_\mu}{2}} O(t^2) = O(t^4|V(\mathcal{T})|) = O(t^5 n),$$

$$C_1^{\mathrm{time}}, C_2^{\mathrm{time}} = \sum_{\mu\in V(\mathcal{T})}\sum_{u,v\in A_\mu}\sum_{\lambda\in\mathrm{Ch}_\mu}\sum_{p,q\in A_\lambda} O\left(\mathrm{ALG}\,(G[X_\mu])\right) = O(t^7|V(\mathcal{T})|) = O(t^8 n),$$

$$C_3^{\mathrm{time}} = \sum_{(\mu,\lambda)\in E(\mathcal{T})}\sum_{u,v\in A_\mu\cup A_\lambda}\sum_{\theta\in\mathrm{Ch}_\mu\setminus\{\lambda\}}\sum_{p,q\in A_\theta} O(t^2) = O(t^7|E(\mathcal{T})|) = O(t^8 n),$$

$$C_4^{\mathrm{time}} = \sum_{\mu\in V(\mathcal{T})}\sum_{\{\lambda,\lambda'\}\in\binom{\mathrm{Ch}_\mu}{2}}\sum_{u,v\in A_\lambda\cup A_{\lambda'}}\sum_{\theta\in\mathrm{Ch}_{\lambda,\lambda'}\setminus\{\lambda\}}\sum_{p,q\in A_\theta} O(t^2) = O(t^9|V(\mathcal{T})|) = O(t^{10} n).$$

Then, we consider the computational complexity of queries when these are precomputed. First, if $\vec{f}_1,\vec{f}_2$ are precomputed, the query can be solved in $O(t^7|V(\mathcal{T})| + \alpha(|V(\mathcal{T})|) + t^6) = O(t^8 n + \alpha(tn))$ time. Next, if $\vec{f}_1,\vec{f}_2,\vec{f}_3$ are precomputed, the query can be solved in $O(t^7 + \alpha(|V(\mathcal{T})|) + t^6) = O(t^7 + \alpha(tn))$ time. Finally, if $\vec{f}_1,\vec{f}_2,\vec{f}_3,\vec{f}_4$ are precomputed, the query can be solved in $O(\alpha(tn) + t^6)$ time.

Here, the t^6 term appearing in each computational time is the time required to perform $O(1)$ times operation $(\oplus)$ to integrate the data structure without using Tree Product Query (for $\widehat{\oplus}$). Of course, the t^6 term could be replaced by $\alpha(tn)$ if a better semigroup could be defined.

These computation complexity shows that the degree of t in each result is larger than that of r in the case of triconnected component decomposition (SPQR tree). The dominant factor of this is that u,v in each $\vec{f}_i(\cdot,u,v)$ can be taken in $O(t^2)$ ways in the tree decomposition, whereas $O(1)$ ways in triconnected component decomposition.

C Algorithm for connected graphs

In this section, we consider the algorithm for solving Beer Path Query for connected graphs. For a given connected graph $G=(V,E)$ with n vertices and m edges, let $C\subseteq V$ be the set of cut vertices, R be the size of the largest the size of the largest triconnected component among all biconnected connected components, and $R_+ = \max\{1,R\}$. The following theorem holds.

▶ **Theorem 13.** *When we construct a data structure in $O(m + nR_+^2 + |C|^2)$ space with preprocessing using $O((m + n \log R_+)R_+^3 + (n + |C|^2)|C|\alpha(m))$ time, we can answer a query in $O(\alpha(|C|))$ time.*

In the following, we describe an outline of the proof of the above theorems. First, we compute the set C and the set of biconnected components $\mathcal{H}$ by using biconnected component decomposition in $O(n + m)$ time. Then, we arbitrarily take $\rho \in \mathcal{H}$, define $\mathcal{T}$ by $V(\mathcal{T}) = \mathcal{H}$ and $\{H, H'\} \in E(\mathcal{T}) \iff V(H) \cap V(H') \neq \emptyset$, and make a tree $\mathcal{T}$ with ρ as the root. Let G_H be the subgraph of G induced by the vertices that appear in the subtree of $\mathcal{T}$ with H as a root. Also, let c_ρ be a specific vertex in $V(\rho)$, and for each $H \in \mathcal{H}$, let c_H be the only cut vertex that H and its parent in $\mathcal{T}$ have in common.

Next, we define the following symbols as well as the mapping to the SPQR tree:

$$\begin{aligned} f_1(H) &\coloneqq \mathrm{d}^{\mathrm{B}}(G_H, c_H, c_H) \ \ (H \in \mathcal{H}), \\ f_2(H, c, c') &\coloneqq \mathrm{d}^{\mathrm{B}}(G, c, c') \ \ (H \in \mathcal{H}, c, c' \in C \cap V(H)), \\ f_3(H, v) &\coloneqq \mathrm{d}^{\mathrm{B}}(G_H, v, c_H) \ \ (H \in \mathcal{H}, v \in V(H)), \\ f_4(H, v) &\coloneqq \mathrm{d}^{\mathrm{B}}(G_H, c_H, v) \ \ (H \in \mathcal{H}, v \in V(H)). \end{aligned}$$

We can calculate f_1 from the leaves to the root by the following formula:

$$f_1(H) = \min \left\{ \begin{array}{c} \mathrm{d}^{\mathrm{B}}(H, c_H, c_H) \\ \min_{I \in \mathrm{Ch}_H} \{\mathrm{d}(H, c_H, c_I) + f_1(I) + \mathrm{d}(H, c_I, c_H)\} \end{array} \right\}.$$

Note that when H is a leaf, we do not consider the second line on the right side of this equation. Similarly, for equations appearing below, we do not consider any undefined part of the equation if it occurs. Also, let J be the parent of H if it exists, we can calculate f_2 from the root to the leaves by the following formula:

$$f_2(H, c, c') = \min \left\{ \begin{array}{c} \mathrm{d}^{\mathrm{B}}(H, c, c') \\ \min_{I \in \mathrm{Ch}_H} \{\mathrm{d}(H, c, c_I) + f_1(I) + \mathrm{d}(H, c_I, c')\} \\ \mathrm{d}(H, c, c_H) + f_2(J, c_H, c_H) + \mathrm{d}(H, c_H, c') \end{array} \right\}.$$

Furthermore, we can calculate f_3 by the following formula:

$$f_3(H, v) = \min \left\{ \begin{array}{c} \mathrm{d}^{\mathrm{B}}(H, v, c_H) \\ \min_{I \in \mathrm{Ch}_H} \{\mathrm{d}(H, v, c_I) + f_2(H, c_I, c_H)\} \end{array} \right\}.$$

f_4 can be computed as well as f_3.

Using the above symbols, we represent the distance between two given vertices s,t and the beer distance. First, we choose a biconnected component that contains the vertices s and t, respectively, and denote it by H_s and H_t.

If $H_s = H_t = H$, the distance and the beer distance can be computed as follow:

$$\mathrm{d}(G, s, t) = \mathrm{d}(H, s, t), \ \ \mathrm{d}^{\mathrm{B}}(G, s, t) = \min \left\{ \begin{array}{c} \mathrm{d}^{\mathrm{B}}(H, s, t) \\ \min_{I \in \mathrm{Ch}_H} \{\mathrm{d}(H, s, c_I) + f_1(I) + \mathrm{d}(H, c_I, t)\} \\ \mathrm{d}(H, s, c_H) + f_2(H, c_H, c_H) + \mathrm{d}(H, c_H, t) \end{array} \right\}.$$

If $H_s \neq H_t$, let H_a be the lowest common ancestor of H_s and H_t in $\mathcal{T}$ and denote the H_s-H_t path in $\mathcal{T}$ by the sequence $H_s = I_1, I_2, \ldots, I_{k-1}, I_k, H_a, J_1, J_2, \ldots, J_{\ell-1}, J_\ell = H_t$.

Then, the distance and the beer distance can be computed as follow:

$$\begin{aligned}\mathrm{d}(G,s,t) &= \mathrm{d}(I_1,s,c_{I_1}) + \textstyle\sum_{2\le p\le k}\mathrm{d}\big(I_p,c_{I_{p-1}},c_{I_p}\big)\\ &\quad + \mathrm{d}(H_a,c_{I_k},c_{J_1}) + \textstyle\sum_{1\le q\le \ell-1}\mathrm{d}\big(J_q,c_{J_q},c_{J_{q+1}}\big) + \mathrm{d}(J_\ell,c_{J_\ell},t)\,,\\ \mathrm{d}^{\mathrm{B}}(G,s,t) &= \mathrm{d}(G,u,v) + \min\left\{\begin{array}{c} f_3(H,s)-\mathrm{d}(I_1,s,c_{I_1})\\ \min_{2\le p\le k}\left\{f_2(I_p,c_{I_{p-1}},c_{I_p})-\mathrm{d}\big(I_p,c_{I_{p-1}},c_{I_p}\big)\right\}\\ f_2(H_a,c_{I_{k-1}},c_{J_2})-\mathrm{d}\big(H_a,c_{I_{k-1}},c_{J_2}\big)\\ \min_{1\le q\le \ell-1}\left\{f_2(J_q,c_{J_q},c_{J_{q+1}})-\mathrm{d}\big(J_q,c_{J_q},c_{J_{q+1}}\big)\right\}\\ f_4(H,t)-\mathrm{d}(J_\ell,c_{J_\ell},t)\end{array}\right\}.\end{aligned}$$

From here, we analyze the computation complexity. Note that we assume that each biconnected graph $H\in\mathcal{H}$ has been preprocessed in $O(m+nR^2)$ space and $O((m+n\log R_+)R_+^3)$ time so that any distance and any beer distance can be computed in $O(\alpha(|E(H)|))=O(\alpha(m))$ time. Also, let $\deg_{\mathcal{T}}(H)$ be the degree of H in $\mathcal{T}$, the space C_i^{space} to store f_i and the time C_i^{time} required to compute f_i can be evaluated as follows:

$$C_1^{\mathrm{space}}=O(|C|),\ C_2^{\mathrm{space}}=\sum_{H\in\mathcal{H}}O(\deg_{\mathcal{T}}(H)^2)=O(|C|^2),$$

$$C_3^{\mathrm{space}},C_4^{\mathrm{space}}=\sum_{H\in\mathcal{H}}O\left(\sum_{v\in V(H)\setminus C}1+\sum_{v\in V(H)\cap C}1\right)=O((n-|C|)+2|C|)=O(n),$$

$$C_1^{\mathrm{time}}=\sum_{H\in\mathcal{H}}O(\alpha(m)|\mathrm{Ch}_H|)=O\left(\alpha(m)\sum_{H\in\mathcal{H}}\deg_{\mathcal{T}}(H)\right)=O(\alpha(m)|C|),$$

$$C_2^{\mathrm{time}}=\sum_{H\in\mathcal{H}}\sum_{c,c'\in C\cap V(H)}O(\alpha(m)|\mathrm{Ch}_H|)=O\left(\alpha(m)\sum_{H\in\mathcal{H}}\deg_{\mathcal{T}}(H)^3\right)=O(\alpha(m)|C|^3),$$

$$C_3^{\mathrm{time}},C_4^{\mathrm{time}}=\sum_{H\in\mathcal{H}}\sum_{v\in V(H)}O(\alpha(m)|\mathrm{Ch}_H|)=\sum_{v\in V}O\left(\alpha(m)\sum_{H\in\mathcal{H}}|\mathrm{Ch}_H|\right)=O(\alpha(m)|C|n).$$

Therefore, we can perform all preprocessing in $O((m+n\log R_+)R_+^3+(n+|C|^2)|C|\alpha(m))$ time and store it in $O(m+nR_+^2+|C|^2)$ space. Furthermore, By using Tree Product Query and Lange Minimum Query, the query can be solved in $O(\alpha(|V(\mathcal{T})|))=O(\alpha(|C|))$ time.

D Figures

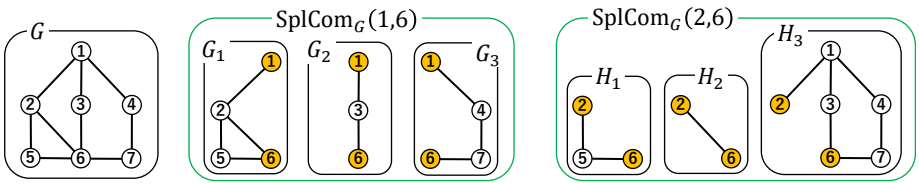


Figure 6 An graph G and its two sets of split components.

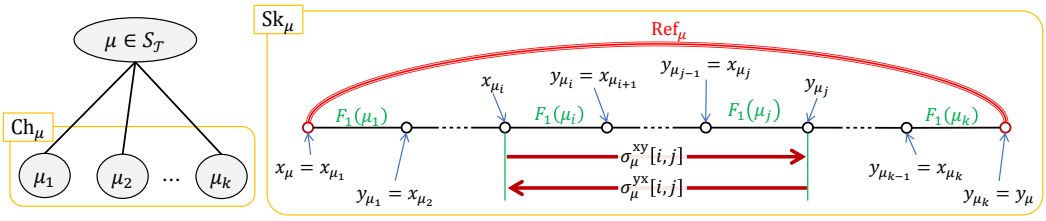


Figure 7 Notation for skeleton of S node.

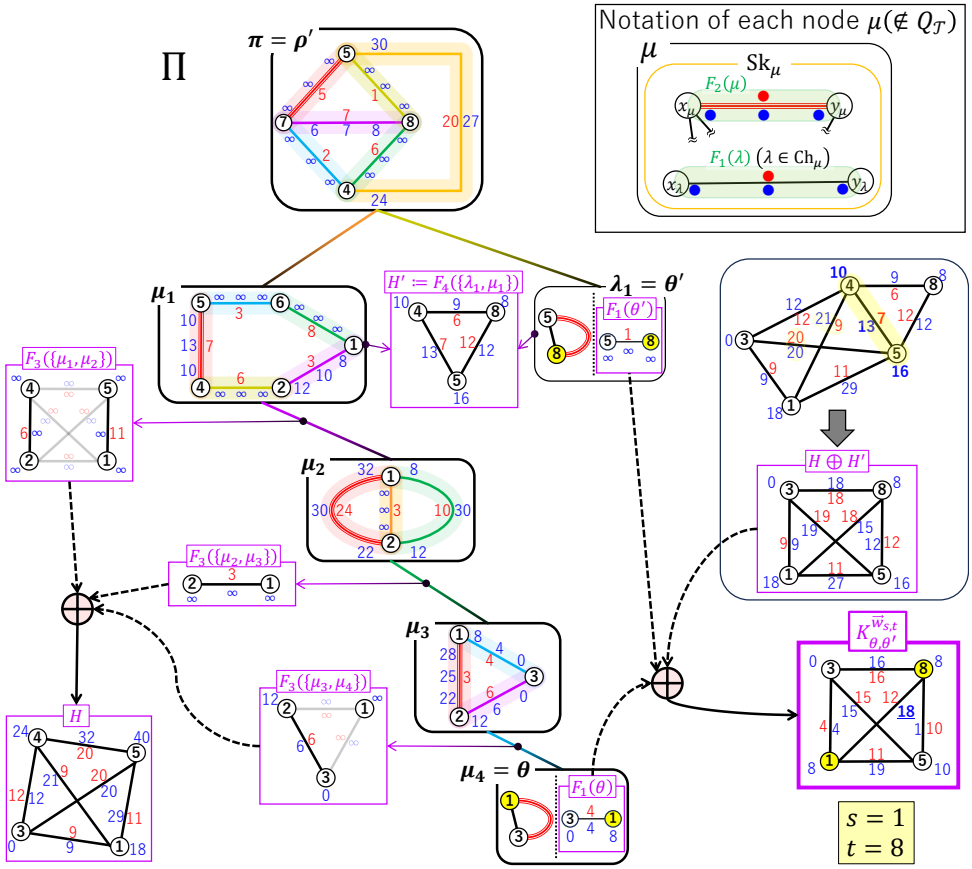


Figure 8 F_3, F_4 on Π and calculation of the beer distance between $s = 1$ and $t = 8$.

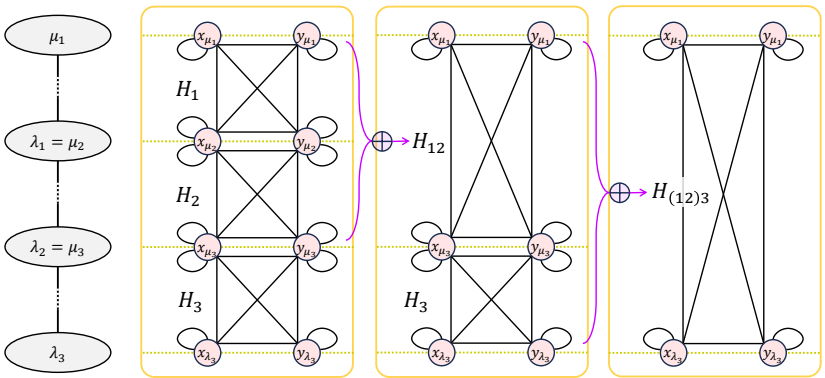


Figure 9 Calculation of $H_{(12)3}$ in the case of $\lambda_1 = \mu_2$ and $\lambda_2 = \mu_3$.

Temporal Separators with Deadlines

Hovhannes A. Harutyunyan
Department of Computer Science and Software Engineering,
Concordia University, Montreal, Canada

Kamran Koupayi
Department of Computer Science and Software Engineering,
Concordia University, Montreal, Canada

Denis Pankratov
Department of Computer Science and Software Engineering,
Concordia University, Montreal, Canada

Abstract

We study temporal analogues of the Unrestricted Vertex Separator problem from the static world. An (s, z)-temporal separator is a set of vertices whose removal disconnects vertex s from vertex z for every time step in a temporal graph. The (s, z)-Temporal Separator problem asks to find the minimum size of an (s, z)-temporal separator for the given temporal graph. The (s, z)-Temporal Separator problem is known to be $\mathcal{NP}$-hard in general, although some special cases (such as bounded treewidth) admit efficient algorithms [15].

We introduce a generalization of this problem called the (s, z, t)-Temporal Separator problem, where the goal is to find a smallest subset of vertices whose removal eliminates all temporal paths from s to z which take less than t time steps. Let τ denote the number of time steps over which the temporal graph is defined (we consider discrete time steps). We characterize the set of parameters τ and t when the problem is $\mathcal{NP}$-hard and when it is polynomial time solvable. Then we present a τ-approximation algorithm for the (s, z)-Temporal Separator problem and convert it to a τ^2-approximation algorithm for the (s, z, t)-Temporal Separator problem. We also present an inapproximability lower bound of $\Omega(\ln(n) + \ln(\tau))$ for the (s, z, t)-Temporal Separator problem assuming that $\mathcal{NP} \not\subset \textsc{Dtime}(n^{\log \log n})$. Then we consider three special families of graphs: (1) graphs of branchwidth at most 2, (2) graphs G such that the removal of s and z leaves a tree, and (3) graphs of bounded pathwidth. We present polynomial-time algorithms to find a minimum (s, z, t)-temporal separator for (1) and (2). As for (3), we show a polynomial-time reduction from the Discrete Segment Covering problem with bounded-length segments to the (s, z, t)-Temporal Separator problem where the temporal graph has bounded pathwidth.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Temporal graphs, dynamic graphs, vertex separator, vertex cut, separating set, deadlines, inapproximability, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.38

Related Version *Full Version*: `https://arxiv.org/abs/2309.14185` [17]

Funding The research presented in this work is supported by NSERC.

1 Introduction

Suppose that you have been given the task of deciding how robust a train system of a given city is with respect to station closures. For instance, is it possible to disconnect the two most visited places, e.g., the downtown and the beach, by shutting down 5 train stations in the city? Does an efficient algorithm even exist? If not, what can we say about special classes of graphs? These are central questions of interest in this work.

More formally, we model the scenario as a graph problem. An important component missing from the classical graph theory is the ability of the graph to vary with time. The trains run on a schedule (or at least they are supposed to – for simplicity, we assume a perfectly punctual train system). Thus, it is not accurate to say that there is an edge between station A and station B just because there are tracks connecting them. It would be more accurate to say that if you arrive at A at some specific time t then you could get to B at some other time $t' > t$, where t is when the train arrives at station A and t' is the time when this train reaches station B. In other words, we can consider the edge from A to B as being present at a particular time (or times) and absent otherwise. This is an important point for the robustness of train networks, since it could be that due to incompatibility of certain train schedules the train network could become disconnected by shutting down even fewer stations than we otherwise would have thought if we didn't take time schedules into account.

The notion of graphs evolving with time has several formal models in the research literature [3, 23]. First of all, there is an area of online algorithms [1] where the graph is revealed piece by piece (thus the only allowable changes are to add objects or relations to the graph) and we need to make irrevocable decisions towards some optimization goal as the graph is being revealed. Secondly, streaming and semi-streaming graph algorithms deal with graphs that are revealed one piece at a time similar to online algorithms, but the emphasis is on memory-limited algorithms [12, 11]. Thus, in streaming one does not have to make irrevocable decisions, but instead tries to minimize the memory size necessary to answer some queries at the end of the stream. Thirdly, there is a notion of dynamic graph algorithms where the emphasis is on designing efficient data structures to support certain queries when the graph is updated by either adding or removing vertices or edges [24]. The goal is to maintain the data structures and answer queries, such as "are nodes u and v connected?", in the presence of changes more efficiently than recomputing the answer from scratch on every query. It is evident that none of these models is a good fit for our question: the train system is known in advance and it is not frequently updated (some cities that shall remain unnamed take decades to add a single station to the system). Fortunately, there is yet another model of graphs changing with time that has recently gotten a lot of attention and it happens to capture our situation perfectly. The model is called a temporal graph. In this work, we focus on undirected temporal graphs that have a fixed node set but whose edge sets change in discrete time units, all of which are known in advance. Other temporal graph models where changes to nodes are allowed and where time is modelled with the continuous real line have been considered in the research literature but they are outside of the scope of this work. We typically use τ to indicate the total number of time steps over which a given temporal graph is defined. For example, if we model the train system as a temporal graph with one minute-granularity and the schedule repeats every 24 hours then the temporal graph would have $\tau = (24H) \times (60M/H) = 1440M$ time steps in total. For emphasis, when we need to talk about non-temporal graphs and bring attention to their unchanging nature we shall call them "static graphs."

We study temporal analogues of the Unrestricted Vertex Separator problem from the static world. An (s, z)-temporal separator is a set of vertices whose removal disconnects vertex s from vertex z for every time step in a temporal graph. The (s, z)-Temporal Separator problem asks to find the minimum size of an (s, z)-temporal separator for the given temporal graph. The (s, z)-Temporal Separator problem is known to be $\mathcal{NP}$-hard in general [28], although some special cases (such as bounded treewidth) admit efficient algorithms [15]. This question can be thought of as a mathematical abstraction of the robustness of the train network of a city question posed at the beginning of this section. The (s, z)-Temporal

Separator problem asks you to eliminate all temporal paths between s and z by removing some nodes. Observe that, practically speaking, in real life, one doesn't actually have to eliminate all temporal paths between s and z – one would have to remove only reasonable temporal paths between s and z. Which paths would be considered unreasonable? We consider paths taking too much time as unreasonable. For example, if normally it takes 30 minutes to get from downtown to the beach, then eliminating all routes that take at most 4 hours would surely detract most downtown dwellers from visiting the beach. Motivated by such considerations, we introduce a generalization of the (s, z)-Temporal Separator problem called (s, z, t)-Temporal Separator problem, where the goal is to find the smallest subset of vertices whose removal eliminates all temporal paths from s to z which takes less than t time steps. Observe that setting $t = \tau$ captures the (s, z)-Temporal Separator problem as a special case of the (s, z, t)-Temporal Separator problem. Our results can be summarized as follows:

In Section 4.1, we present a characterization of parameters t and τ when the problem is $\mathcal{NP}$-hard. We also present an inapproximability lower bound of $\Omega(\ln(n) + \ln(\tau))$ for the (s, z, t)-Temporal Separator problem assuming that $\mathcal{NP} \not\subset \textsc{Dtime}(n^{\log\log n})$. In Section 4.2, we present a τ-approximation algorithm for the (s, z)-Temporal Separator problem, and we convert it to a τ^2-approximation algorithm for (s, z, t)-Temporal Separator problem.

In Section 5.1, we present a polynomial-time algorithm to find a minimum (s, z, t)-temporal separator on temporal graphs whose underlying graph (see Section 2) has branchwidth at most 2. In Section 5.2, we present another polynomial-time algorithm for temporal graphs whose underlying graph becomes a tree after removal of s and z. In Section 5.3, we show a polynomial-time reduction from the Discrete Segment Covering problem with bounded-length segments to the (s, z, t)-Temporal Separator problem where the temporal graph has bounded pathwidth. Therefore, solving the (s, z, t)-Temporal Separator problem on a temporal graph whose underlying graph has bounded pathwidth is at least as difficult as solving the Discrete Segment Covering problem where lengths of all segments are bounded.

2 Preliminaries

Temporal graphs (also known as dynamic, evolving [13], or time-varying [14, 6] graphs) are graphs whose edges are active at certain points in time. A temporal graph $G = (V, E, \tau)$ contains a set of vertices V, and a set of edges $E \subseteq V \times V \times [\tau]$ [1]. So each edge[2]. $e \in E$ contains two vertices of V and a time label $t \in [\tau]$ indicating a time step at which the edge is active. A graph $G_{\downarrow} = (V, E')$ where E' contains every edge e that is active at least once in the temporal graph G is called the *underlying graph* (alternatively, the *footprint*) of the temporal graph G. A static graph representing active edges for a specific time t is called the layer of the temporal graph at that time and is denoted by G_t. Some other ways of modelling temporal graphs could be found in [20]. We refer to $V(G)$ and $E(G)$ as the set of vertices and edges, respectively, of a graph G (either temporal or static). Also for any subset $U \subseteq V(G)$ we refer to the set of all edges in the subgraph induced by U as $E(U)$, and for any node $v \in V$ we use $E(v)$ to denote the set of all edges incident on v. We also use $\tau(G)$ to refer to the number τ of time labels of the temporal graph G.

[1] Notation $[n]$ stands for $\{1, 2, \ldots, n\}$.

[2] We only consider undirected graphs in this work, i.e. no self-loops and $(u, v, t) \in E$ if and only if $(v, u, t) \in E$

A temporal path in a temporal graph is a sequence of edges such that (1) it is a valid path in the underlying graph, and (2) the corresponding sequence of times when the edges are active is non-decreasing. Formally, a sequence $P = [(u_1, v_1, t_1), (u_2, v_2, t_2), \ldots, (u_k, v_k, t_k)]$ of edges in a temporal graph G is called an (s, z)-*temporal path* if $s = u_1, v_1 = u_2, \ldots, v_{k-1} = u_k, v_k = z$ and $t_1 \leq t_2 \leq \cdots \leq t_k$. If the sequence of times is in strictly increasing order, the temporal path is called *strict*. *Travelling time* of P, denoted by $\text{ttime}(P)$, is defined as $\text{ttime}(P) = t_k - t_1 + 1$, i.e., the time it takes to travel from s to z. If $\text{ttime}(P) \leq t$ then we refer to P as an (s, z, t)-temporal path. A temporal graph G is *connected* if for any pair of vertices $s, z \in V(G)$ there is at least one temporal path from s to z. A temporal graph G is *continuously connected* if for every $i \in [\tau(G)]$ layer G_i is connected.

We distinguish between three types of temporal paths: (1) *shortest* (s, z)*-temporal path*: a temporal path from s to z that minimizes the number of edges; (2) *fastest* (s, z)*-temporal path*: a temporal path from s to z that minimizes the traveling time; (3) *foremost* (s, z)*-temporal path*: a temporal path from s to z that minimizes the arrival time at destination. *Temporal distance* from node s to node z is equal to the traveling time of the fastest (s, z)-temporal path.

A set $S \subseteq V - \{s, z\}$ is called a *(strict)* (s, z)*-temporal separator* if the removal of vertices in set S removes all (strict) temporal paths from s to z. The *(strict)* (s, z)*-Temporal Separator problem* asks to find the minimum size of a (strict) (s, z)-temporal separator in a given temporal graph G. This problem has been studied before (see Section 3). In this work, we propose a new problem that is based on the notion of (s, z, t)-temporal paths. We define a set of vertices S to be a *(strict)* (s, z, t)*-temporal separator* if every (strict) (s, z, t)-temporal path contains at least one vertex in S, i.e., removal of S removes all (strict) (s, z, t)-temporal paths. Thus, the new problem, which we refer to as the *(strict)* (s, z, t)*-Temporal Separator problem* is defined as follows: given a temporal graph G, a pair of vertices $s, z \in V(G)$, and a positive integer t, the goal is to compute the minimum size of a (s, z, t)-temporal separator in G.

▶ **Lemma 1.** *Given a temporal graph $G = (V, E, \tau)$ and two distinct vertices s and z as well as an integer t, it is decidable in time $O(|S||E|)$ if there is a (s, z, t)-temporal path in G where $S = \{t' \mid \exists u : (s, u, t') \in E\}$.*

Proof. [25] and [27] present an algorithm that computes fastest paths from a single source s to all of the vertices in $O(|S|(|V| + |E|))$. We could ignore isolated vertices, then we could compute a fastest path from s to z in G and check if its travelling time is at least t. ◀

Branch decomposition and branchwidth of a graph is defined as follows.

▶ **Definition 2** (Branch Decomposition [8])**.** *Given a graph $G = (V, E)$, a branch decomposition is a pair (T, β), such that*

- *T is a binary tree with $|E|$ leaves, and every inner node of T has two children.*
- *β is a mapping from $V(T)$ to 2^E satisfying the following conditions:*
  - *For each leaf $v \in V(T)$, there exists $e \in E(G)$ with $\beta(v) = \{e\}$, and there are no $v, u \in V(T), v \neq u$ such that $\beta(v) = \beta(u)$.*
  - *For every inner node $v \in V(T)$ with children v_l, v_r, $\beta(v) = \beta(v_l) \cup \beta(v_r)$;*

▶ **Definition 3** (Boundary [8])**.** *Given a graph $G = (V, E)$, for every set $F \subseteq E$, the boundary $\partial F = \{v | v$ is incident to edges in F and $E \backslash F\}$.*

▶ **Definition 4** (Width of a Branch Decomposition [8])**.** *Given a branch decomposition (T, β) of $G = (V, E)$, the width of this decomposition is $max\{|\partial\beta(v)| \mid v \in V(T)\}$.*

The branchwidth $bw(G)$ of G is defined as the minimum width of a branch decomposition of G [8]. We note that for any fixed k there is a linear time algorithm to check if a graph has branchwidth k, and if so, the algorithm outputs a branch decomposition of minimum width [5].

Path decomposition and pathwidth of a graph are defined as follows.

▶ **Definition 5** (Path Decomposition [22]). *Given a graph $G = (V, E)$, a path decomposition of G is a pair (P, β), such that*

- *P is a path with nodes $a_1, \dots a_m$.*
- *β is a mapping from $\{a_1, \dots, a_m\}$ to 2^E satisfying the following conditions:*
  - *For $e \in E(G)$ there exists a_i such that vertices of e appear in $\beta(a_i)$.*
  - *For every $v \in V(G)$ the set of a_i, such that v appears in $\beta(a_i)$, forms a subpath of P.*

The width of a decomposition (P, β) is $\max_{a \in V(P)} |\beta(a)| - 1$. The pathwidth of a graph G is the minimum width of a path decomposition of G.

3 Related Work

Enright et al. in [9] adopt a simple and natural model for time-varying networks which is given with time-labels on the edges of a graph, while the vertex set remains unchanged. This formalism originates in the foundational work of Kempe et al. [18]. There has already been a lot of work on temporal graphs, too much to give a full overview. Thus, in this section, we focus only on the results most relevant to our work.

The fastest temporal path is computable in polynomial time, see, e.g. [27, 26, 25]. A nice property of the foremost temporal path is that it can be computed efficiently. In particular, there is an algorithm that, given a source node $s \in V$ and a time t_{start}, computes for all $w \in V \setminus \{s\}$ a foremost (s, w)-temporal path from the time t_{start} [19]. The running time of the algorithm is $O(n\tau^3 + |E|)$. It is worth mentioning that this algorithm takes as input the whole temporal graph G. Such algorithms are known as offline algorithms in contrast to online algorithms in which the temporal graph is revealed on the fly. The algorithm is essentially a temporal translation of the breadth-first search (BFS) algorithm (see e.g. [7] page 531).

While the Unrestricted Vertex Separator problem is polynomial time solvable in the static graph world (by reducing to the Maximum Flow problem), the analogous problem in the temporal graph world, namely, the (s, z)-Temporal Separator problem, was shown to be $\mathcal{NP}$-hard by Kempe et al. [18]. Zschoche et al. [28] investigate the (s, z)-Temporal Separator and strict (s, z)-Temporal Separator problems on different types of temporal graphs. A central contribution in [28] is to prove that both $(s, z)-$Temporal Separator and Strict (s, z)-Temporal Separator are $\mathcal{NP}$-hard for all $\tau \geq 2$ and $\tau \geq 5$, respectively, strengthening a result by Kempe et al. [18] (they show $\mathcal{NP}$-hardness of both variants for all $\tau \geq 12$) [28].

Fluschnik et al. [15] show that (s, z)-Temporal Separator remains $\mathcal{NP}$-hard on many restricted temporal graph classes: temporal graphs whose underlying graph falls into a class of graphs containing complete-but-one graphs (that is, complete graphs where exactly one edge is missing), or line graphs, or temporal graphs where each layer contains only one edge. In contrast, the problem is tractable if the underlying graph has bounded treewidth, or if we require each layer to be a unit interval graph and impose suitable restrictions on how the intervals may change over time, or if one layer contains all others (grounded), or if all layers are identical (1-periodic or 0-steady), or if the number of periods is at least the number of vertices. It is not difficult to show that this problem is fixed-parameter tractable when parameterized by $k + l$, where k is the solution size and l is the maximum length of a temporal (s, z)-path.

Lastly, we note that the classical Vertex Separator problem from the static world is often stated as asking to find a vertex separator such that after its removal the graph is partitioned into two blocks (one containing s and one containing z) of roughly equal size[3]. This "balanced" separator restriction makes the problem $\mathcal{NP}$-hard. The temporal separator problems considered in this work do not have such a restriction, and as discussed they are hard problems due to the temporal component. There is a lot of research on the Vertex Separator problem, but since our versions do not have this "balancedness" restriction, we do not discuss it in detail. An interested reader is referred to [2] and references therein.

4 Temporal Separators with Deadlines on General Graphs

4.1 Hardness of Exact and Approximate Solutions

Zschoche et al. [28] show that the (s, z)-Temporal Separator problem is $\mathcal{NP}$-hard on a temporal graph $G = (V, E, \tau)$ if $\tau \geq 2$ (and it is in $\mathcal{P}$ if $\tau = 1$). So, it is obvious that the (s, z, t)-Temporal Separator problem is $\mathcal{NP}$-hard if $t \geq 2$. In this section we strengthen this result by showing that the problem remains $\mathcal{NP}$-hard even when restricted to inputs with $t = 1$ and $\tau \geq 2$.

Reduction from the minimum satisfiability problem with non-negative variables to $(s, z, 1)$-Temporal Separator could be made by adding a path from s to z in layer G_i, which contains all the variables in the i-th clause. So, $(s, z, 1)$-Temporal Separator on temporal graphs with a sufficient number of layers is $\mathcal{NP}$-hard. However, it is not easy to establish the complexity of (s, z, t)-Temporal Separator on temporal graphs with a small number of layers. Here we aim to show that $(s, z, 1)$-Temporal Separator remains $\mathcal{NP}$-hard on a temporal graph $G = (V, E, \tau)$ if τ is equal to 2. To do that, we construct a reduction from the Node Multiway Cut problem. In this problem, one is given a graph $G = (V, E)$ and a set of terminal vertices $Z = \{z_1, z_2, \ldots z_k\}$. A multiway cut $S \in V \backslash Z$ is a set of vertices whose removal from G disconnects all pairs of distinct terminals z_i and z_j. The goal is to find a multiway cut of minimum cardinality. The Node Multiway Cut problem is $\mathcal{NP}$-hard for $k \geq 3$ [16]. For the proof of the next theorem, please see the full version of the paper [17].

▶ **Theorem 6.** *For every $t_0 \geq 1$, the (s, z, t)-Temporal Separator problem is $\mathcal{NP}$-hard on a temporal graph $G = (V, E, \tau)$ when restricted to inputs with $t = t_0$ and $\tau \geq 2$.*

Since Strict (s, z)-Temporal Separator is $\mathcal{NP}$-hard on a temporal graph with $\tau \geq 5$ [28], it is clear that Strict (s, z, t)-Temporal Separator is $\mathcal{NP}$-hard even when restricted to inputs with $t \geq 5$ and $\tau \geq 5$. However, by a small change to the reduction presented by Zschoche et al. [28], which is inspired by [26], we can show that Strict (s, z, t)-Temporal Separator remains $\mathcal{NP}$-hard even when restricted to inputs with $t = 3$ and $\tau = 4$. For the proof of the next theorem, please see the full version of the paper [17].

▶ **Theorem 7.** *Finding a strict $(s, z, 3)$-temporal separator on a temporal graph $G = (V, E, \tau)$ is $\mathcal{NP}$-hard when restricted to inputs with $\tau = 4$.*

Since every temporal path from s to z contains more than two edges, then $\emptyset$ is a strict $(s, z, 1)$-temporal separator. Since every strict $(s, z, 2)$-temporal path is of the form $(s, v, t), (v, z, t + 1)$, the Strict $(s, z, 2)$-Temporal Separator problem could be solved in

[3] That is why earlier we referred to a static world problem of interest as the Unrestricted Vertex Separator problem to emphasize that there is no balancedness requirement.

polynomial time easily. The Strict (s, z, t)-Temporal Separator problem on a graph $G = (V, E, \tau)$ with $\tau = t$ is the same as the Strict (s, z)-Temporal Separator. Therefore, in case $\tau = t = 3$ this problem is equivalent to the Strict (s, z)-Temporal Separator problem with $\tau = 3$. Zschoche et al. [28] present a polynomial time algorithm for finding a minimum strict (s, z)-temporal separator on a temporal graph $G = (V, E, \tau)$ when $\tau < 5$. So, this case could be solved in polynomial time. Although we know that finding a strict (s, z, t)-temporal separator on a temporal graph $G = (V, E, 3)$ is polynomial-time solvable with the algorithm which is presented in [28], we describe another simple algorithm to solve this problem.

In the first step of the algorithm, we check if there is an edge between s and t. If so, it is clear that there are no separator sets because the direct path using this edge from s to z will remain with the removal of any node from the graph.

Next, for every temporal path from s to z of length two, such as $(s, x, t_1), (x, z\ , t_2)$ with $t_2 = t_1 + 1$, it is clear that we have to remove x if we want to remove this path from the graph. So, it is clear that $x \in S$.

In the last step, we know that the length of every temporal path in the graph is three. So, every path from s to z should be of the following form:

$$(s, x, 1), (x, y, 2), (y, z, 3).$$

Now, put every node x with existing edge (s, x) into the set X with time label 1. Also, put every node y that is a neighbor of z into the set Y with time label 3. Now, it is clear that $X \cap Y = \emptyset$, for otherwise there exists a node u with two existing edges $e_1 = (s, u, 1)$ and $e_2 = (u, z, 3)$, while this node should be removed in the previous step. Therefore, every strict temporal path from s to z should have a corresponding edge $(x, y, 2)$ where $x \in X$ and $y \in Y$. So, we should remove either x or y for every edge $(x, y, 2)$, where $x \in X$ and $y \in Y$. In order to do this we could use any known polynomial time algorithm for the Vertex Cover problem in bipartite graphs.

In the rest of this section we show $\Omega(\log n + \log(\tau))$-inapproximability (assuming $\mathcal{NP} \subset \text{DTIME}(n^{\log \log n})$) for the (s, z, t)-Temporal Separator problem. This is proved by a strict reduction from the Set Cover problem. Recall that in the Set Cover problem, one is given a collection $\mathcal{S}$ of subsets of a universe U that jointly cover the universe. The goal is to find a minimum size sub-collection of $\mathcal{S}$ that covers U.

▶ **Theorem 8.** *For every $t > 0$ there is a strict polynomial time reduction from the Set Cover problem to the $(s, z, t)-$Temporal Separator problem.*

Proof. Let $(U, \mathcal{S})$ be an instance of the Set Cover problem, where $U = \{1, 2, \dots n\}$ is the universe and $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$ is a family of sets the union of which covers U. For each $i \in U$ define the family $\mathcal{F}_i$ as $\mathcal{F}_i = \{S \in \mathcal{S} \mid i \in S\}$, i.e., $\mathcal{F}_i$ consists of all sets from $\mathcal{S}$ that contain element i. Let $k_i = |\mathcal{F}_i|$ and order the elements of each $\mathcal{F}_i$ in the order of increasing indices, i.e., $\mathcal{F}_i = \{S_{i_1}, \dots, S_{i_{k_i}}\}$.

Our reduction outputs a temporal graph $f(U, \mathcal{S}) = (V \cup \{s, z\}, E)$ where:

- the vertex set is $V \cup \{s, z\} = \{v_i | i \in [m]\} \cup \{s, z\}$;
- the edge set is the union over i of the sets $E_i = \{(s, v_{i_1}, i{\cdot}t), (v_{i_1}, v_{i_2}, i{\cdot}t), \dots, (v_{i_{k_i-1}}, v_{i_{k_i}}, i{\cdot}t), (v_{i_{k_i}}, z, i \cdot t)\}$.

The main idea behind the proof is to map every element of U to a path from s to z in $f(U, \mathcal{S})$ bijectively, so by covering an element, we remove the corresponding path in $f(U, \mathcal{S})$ as well as by removing a path we cover the corresponding element.

We claim that $V' = \{v_{j_1}, \dots, v_{j_\ell}\} \subseteq V$ is a $(s, z, t)-$temporal separator for $f(U, \mathcal{S})$ if and only if $\mathcal{S}' = \{S_{j_1}, \dots, S_{j_\ell}\} \subseteq \mathcal{S}$ is a set cover for $(U, \mathcal{S})$.

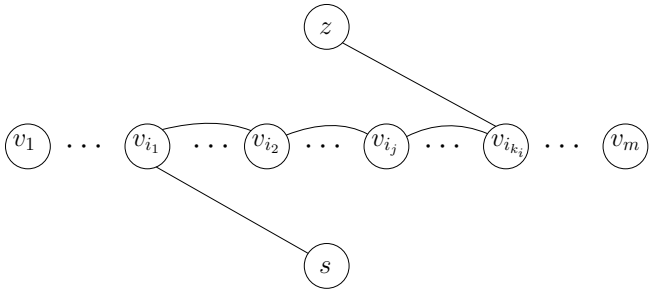


Figure 1 Layer $G_{i\cdot t}$ of the temporal graph used in the proof of Theorem 8.

Figure 1 represents the edges in the layer G_{it}, which contain all the edges in E_i. It illustrates that element i in the universe U corresponds to a path E_i, as well as the element i is covered by the set $S_{i_j} \in \mathcal{S}'$ if and only if a temporal path which is shown in Figure 1 is removed from the temporal graph by removing the vertex $v_{i_j} \in V'$.

$\rightarrow$ Suppose for contradiction that $\mathcal{S}'$ does not cover U. Pick an arbitrary item $i \in U$ that is not covered and consider the following path $P = [(s, v_{i_1}, i \cdot t), (v_{i_1}, v_{i_2}, i \cdot t), \ldots, (v_{i_{k_i-1}}, v_{i_{k_i}}, i \cdot t), (v_{i_{k_i}}, z, i \cdot t)]$, where the indices are according to the equation for $\mathcal{F}_i$. Since i is not covered, $\mathcal{F}_i \cap \mathcal{S}' = \emptyset$, so P is present in $f(U, \mathcal{S}) \setminus V'$ violating the assumption that V' is a $(s, z, t)-$temporal separator (note that $\text{ttime}(P) = 0$).

$\leftarrow$ Now, suppose for contradiction that V' is not a (s, z, t)-temporal separator. Thus, there is path P from s to z with $\text{ttime}(P) < t$. From the definition of $f(U, \mathcal{S})$ it is clear that P should be using edges only from E_j for some $j \in [n]$. Note that there is a unique (s, z)-temporal path that can be constructed from E_j, namely, $P = [(s, v_{j_1}, j \cdot t), (v_{j_1}, v_{j_2}, j \cdot t), \ldots, (v_{j_{k_j-1}}, v_{j_{k_j}}, j \cdot t), (v_{j_{k_j}}, z, j \cdot t)]$. This implies that element j is not covered by $\mathcal{S}'$, since otherwise, one of the v_{j_i} would be in V'.

Following the previous claim, every solution in (s, z, t)-Temporal Separator has a corresponding solution in Set Cover, and vice versa. Therefore, an optimal solution in (s, z, t)-Temporal Separator, has a corresponding optimal solution in Set Cover. As a result $\frac{|V'|}{|V_{opt}|} = \frac{|S'|}{|S_{opt}|}$. This implies that the reduction is strict. ◀

Due to the inapproximability of Set Cover (see [10]), we have the following:

▶ **Corollary 9.** *The (s, z, t)-Temporal Separator problem is not approximable to within $(1 - \epsilon)(\log n + \log(\tau))$ in polynomial time for any $\varepsilon > 0$, unless $\mathcal{NP} \subset \text{DTIME}(n^{\log \log n})$.*

4.2 Approximation Algorithms

In this section, we present an efficient τ^2-approximation for the (s, z, t)-Temporal Separator problem. We begin by establishing a τ-approximation for the (s, z)-Temporal Separator problem. The main tool used in this section is the *flattening*[4] of a temporal graph $G = (V, E, \tau)$ with respect to vertices s and z, denoted by $F(G, s, z) = (V', E')$. To ease the notation we omit the specification of s and z and denote the flattening of G by $F(G)$. The flattening $F(G)$ is a static directed graph defined as follows: the vertex set V' is the union of τ disjoint sets $V_1, V_2, \ldots, V_\tau$ and $\{s, z\}$, where each V_i is a disjoint copy of $V - \{s, z\}$. Denoting the

[4] The concept of flattening is not new, and it is similar to the static expansion of a temporal graph – see, for example, [19].

vertices of V by $v_1, v_2, \ldots, v_n$, we have $\forall i \in [\tau]\ \ V_i = \{v_{j,i} | v_j \in V - \{s, z\}\}$. The edge set E' of the flattening $F(G)$ is defined as follows:

- For each $(v_i, v_j, t') \in E$ with $v_i, v_j \notin \{s, z\}$ we add edges $(v_{i,t'}, v_{j,t'})$ and $(v_{j,t'}, v_{i,t'})$ to E'.
- For each $v_i \in V$ and each time $t' \in [\tau - 1]$ we add an edge $(v_{i,t'}, v_{i,t'+1})$ to E'.
- For each $(s, v_i, t') \in E$ we add an edge $(s, v_{i,t'})$ to E'.
- For each (z, v_i, t') we add an edge $(v_{i,t'}, z)$ to E'.

Clearly, $F(G)$ is defined to express temporal (s, z)-paths in G in terms of (s, z)-paths in $F(G)$. More specifically, if we have a temporal (s, z) path P in G then there is an analogous static (s, z) path P' in $F(G)$. If P begins with an edge (s, v_i, t_1) then P' begins with an edge (s, v_{i,t_1}). After that if the next edge in P is (v_i, v_j, t_2), we can simulate it in $F(G)$ by introducing a sequence of edges $(v_{i,t_1}, v_{i,t_1+1}), (v_{i,t_1+1}, v_{i,t_1+2}), \ldots, (v_{i,t_2-1}, v_{i,t_2})$ followed by an edge (v_{i,t_2}, v_{j,t_2}), and so on until the vertex z is reached. This correspondence works in reverse as well. If P' is a static (s, z) path in $F(G)$ then we can find an equivalent temporal (s, z) path in G as follows. If the first edge in P' is (s, v_{i,t_1}) then this corresponds to the first edge of P being (s, v_i, t_1). For the following edges of P', if the edge is of the form $(v_{i,t'}, v_{i,t'+1})$ then it is simply ignored for the purpose of constructing P (since it corresponds to the scenario where the agent travelling along the path is simply waiting an extra time unit at node v_i), and if the edge is of the form $(v_{i,t'}, v_{j,t'})$ then we add the edge (v_i, v_j, t') to P. This continues until z is reached. Thus, there is a temporal (s, z) path P in G if and only if there is a static (s, z) path P' in $F(G)$. Moreover, if S represents the internal nodes of the path P then we can find P' with internal nodes in $\bigcup_{t' \in [\tau]} \{v_{i,t'} : v_i \in S\}$. In the reverse direction, if P' uses internal nodes S' then we can find P with internal nodes in $\{v_i : \exists t'\ \ v_{i,t'} \in S'\}$.

Armed with these observations, we show that the sizes of (s, z)-temporal separators in G and (s, z)-separators (non-temporal) in $F(G)$ are related as follows.

▶ **Theorem 10.**

1. *If S is an (s, z)-temporal separator in G then there is an (s, z)-separator of size at most $\tau|S|$ in $F(G)$.*
2. *If S' is an (s, z)-separator in $F(G)$ then there is an (s, z)-temporal separator of size at most $|S'|$ in G.*

The proof of the above theorem, albeit rather simple, appears in the full version of the paper [17].

▶ **Corollary 11.** *The (s, z)-Temporal Separator problem on a temporal graph $G = (V, E, \tau)$ can be approximated within τ in $O((m + n\tau)n\tau)$ time, where $n = |V|$ and $m = |E|$.*

Proof. We can use any existing efficient algorithm to solve the (s, z) separator problem on $F(G)$ and return its answer, which will give τ-approximation by Theorem 10. For example, the stated runtime is achieved by applying Menger's theorem and the Ford-Fulkerson algorithm to compute the maximum number of vertex-disjoint paths in $F(G)$. Then the running time is $O(|E'||V'|)$. Observing that $|E'| \leq |E| + |V|\tau$ and $|V'| \leq |V|\tau$, finishes the proof of this corollary. ◀

Next, we describe how the (s, z, t)-Temporal Separator problem can be approximated using a slight extension of the above ideas. First, for a temporal graph $G = (V, E, \tau)$ and two integers $t_1 \leq t_2$ we define $E[t_1 : t_2] = \{(u, v, t) \in E : t_1 \leq t' \leq t_2\}$. We also define $G[t_1 : t_2] = (V, E[t_1 : t_2], t_2)$, which can be thought of as graph G restricted to time interval $[t_1, t_2]$. The idea behind approximating a minimum (s, z, t)-temporal separator is to combine (s, z)-temporal separators of $F(G[1 : t + 1]), F(G[2 : t + 2]), \ldots, F(G[\tau - t : \tau])$.

▶ **Theorem 12.** *The (s, z, t)-Temporal Separator problem on a temporal graph $G = (V, E, \tau)$ can be approximated within τ^2 in $O((m + n\tau)n\tau^2)$ time, where $n = |V|$ and $m = |E|$.*

Proof. The algorithm has essentially been described prior to the statement of the theorem, so the running time is clear. It is left to argue that it produces τ^2-approximation. This can be argued similarly to Theorem 10.

1. Let S be a (s, z, t)-temporal separator in G. Then for $G[i : i + t]$ we define S_i to consist of all nodes $v_{j,t'}$ with $v_j \in S$. Since S removes all paths from G of travelling time $\leq t$ and $G[i : i + t]$ only has paths of travelling time $\leq t$, then S_i is a (s, z)-separator in $G[i : i + t]$ of size $|S_i| = \tau|S|$. Thus, if there is an (s, z, t)-temporal separator of size $|S|$ in G then the combined size of all (s, z, t)-temporal separators of $G[1 : t + 1], G[1 : t + 2], \ldots, G[\tau - t, \tau]$ is at most $\tau^2|S|$.
2. Let S_i be a (s, z)-temporal separator in $G[i : i + t]$. Define $S = \{v_j : \exists i \exists t' \ \ v_{j,t'} \in S_i\}$. It is easy to see that S is a (s, z, t) temporal separator in G. Paths of travelling time at most t that begin with an edge (s, v_i, t_1) are present in $G[t_1, t_1 + t]$, and so removal of S_{t_1} removes such temporal paths in $G[t_1, t_1 + t]$. Since S_{t_1} is "projected" onto V and included in S, these paths are eliminated from G. ◀

5 Temporal Separators with Deadlines on Special Families of Graphs

5.1 Temporal Graphs with Branchwidth at most 2

The graphs with branchwidth 2 are graphs in which each biconnected component is a series-parallel graph [21]. In this section, we present an efficient algorithm to solve the (s, z, t)-Temporal Separator problem on temporal graphs whose underlying static graphs have branchwidth at most 2. In fact, our algorithm works for a more general class of problems, which we refer to as "restricted path (s, z)-Temporal Separator." The goal in this more general problem is to select a set of vertices S such that the removal of S from the given temporal graph G removes all (s, z) paths in a restricted family of paths. The (s, z, t)-Temporal Separator problem is seen as a special case of this, where paths are restricted to have travelling time less than t. Restricted family of paths could be any path family implicitly defined by a procedure $ExistsRestrictedPath(G, s, z)$ which takes as input a temporal graph G, a pair of nodes s and z, and returns true if and only if there exists a restricted temporal path between s and z in G. Due to Lemma 1, we know that such a procedure exists in the case of temporal paths restricted by travelling time, which is suitable for the (s, z, t)-Temporal Separator problem.

For the rest of this section, we assume that G is a temporal graph such that $bw(G_\downarrow) \leq 2$ unless stated otherwise. Furthermore, we assume that $G_\downarrow$ is connected, otherwise, if s and z belong to different connected components the answer to the problem is trivially $\emptyset$, and if they belong to the same connected component, the problem reduces to analyzing that connected component alone. We introduce some notation and make several observations about branch decomposition before we give full details of our algorithm. Recall from Section 2 that branch decomposition of G of width 2 can be computed in linear time. Thus, we assume that the algorithm has access to such a decomposition, which we denote by (T, β). We use ρ to denote the root of T and we define the function $top : V(G) \to V(T)$ as follows. For $v \in V(G)$ we let $top(v)$ be the furthest node $x \in V(T)$ from the root r which satisfies $E(v) \subseteq \beta(x)$. We also use x_l to denote the left child of x and x_r to denote the right child of x. For a node $x \in V(T)$ we define G_x^{in} to be the temporal graph obtained from G by keeping only those edges (u, v, t) with $(u, v) \in \beta(x)$ and removing all vertices of degree 0. We collect several useful observations about the introduced notions in the following lemma.

▶ **Lemma 13.**

1. *If* $v \in \partial\beta(x)$ *then* $v \in \partial\beta(x_\ell)$ *or* $v \in \partial\beta(x_r)$.
2. *If* $top(v) = x$ *then* $v \in \partial\beta(x_\ell)$ *and* $v \in \partial\beta(x_r)$.
3. *If* $v \in V(G_x^{in}) \setminus \partial\beta(x)$ *then all edges incident on* v *in* G *are present in* G_x^{in}.

Proof.

1. Since $v \in \partial\beta(x)$ it means that some but not all edges incident on v in G appear in $\beta(x)$. Since $\beta(x) = \beta(x_\ell) \cup \beta(x_r)$, it implies that some but not all edges incident on v must appear either in $\beta(x_\ell)$, or $\beta(x_r)$, or both.
2. If $top(v) = x$ then $E(v) \subseteq \beta(x)$. Suppose for contradiction that $v \notin \partial\beta(x_\ell)$. This can happen for two reasons: either (1) $E(v) \subseteq \beta(x_\ell)$, or (2) $E(v) \cap \beta(x_\ell) = \emptyset$. In case (1) we obtain a contradiction with the definition of $top(v)$ since x_ℓ is further from the root than x and it still contains all of $E(v)$. In case (2) observe that we must have $E(v) \subseteq \beta(x_r)$, thus obtaining a contradiction with the definition of $top(v)$ again since x_r is further from the root than x and it still contains all of $E(v)$.
3. Since $v \in V(G_x^{in}) \setminus \partial\beta(x)$ it means that there is at least one edge incident on v in $V(G_x^{in})$. Since v is not in the boundary of $\beta(x)$, it means that all edges incident on v in G must be present in $\beta(x)$. ◀

Algorithm 1 This algorithm finds a restricted $(s, z)-$temporal separator in a temporal graph G with $bw(G_\downarrow) \leq 2$.

```
Function RTS(G, s, z):
    if ExistsRestrictedPath(G, s, z) = false then
        return ∅;
    for v ∈ V(G) \ {s, z} do
        if ExistsRestrictedPath(G \ {v}, s, z) = false then
            return {v};
    if top(s) = top(z) then
        return RTS(G^in_{ρ_ℓ}, s, z) ∪ RTS(G^in_{ρ_r}, s, z);
    else if top(s), top(z) are not ancestors of each other then
        return ∂β(top(z));
    else
        /* assume top(z) is ancestor of top(s), otherwise swap s and z   */
        if z ∉ ∂β(top(s)) then
            return ∂β(top(s));
        else if ∂β(top(s)) = {z} then
            return RTS(G^in_{top(s)}, s, z);
        else
            /* ∂β(top(s)) = {z, q}, ∂β(top(s)_ℓ) = {s, z}, ∂β(top(s)_r) = {s, q}   */
            S ← RTS(G^in_{top(s)_ℓ}, s, z);
            if ExistsRestrictedPath(G \ S, s, z) then
                return S ∪ {q};
            else
                return S;
```

Now, we are ready to describe our algorithm, which is denoted by RTS. The algorithm starts by checking if there is a restricted temporal path from s to z in G, and if such a path does not exist then the algorithm immediately returns $\emptyset$. Then the algorithm checks if

there exists a restricted temporal separator of size 1 by testing whether there is a restricted temporal path in $G \setminus \{v\}$ for each $v \in V(G) \setminus \{s, z\}$. Then the algorithm computes $top(s)$ and $top(z)$ and the computation splits into three cases: (1) if $top(s) = top(z)$; (2) if $top(s)$ and $top(z)$ are not on the same root-to-leaf path in T (i.e., neither one is an ancestor of another); and (3) if one of $top(s), top(z)$ is an ancestor of another. We shall later see that case (1) implies that $top(s) = top(z) = \rho$. In this case, the algorithm invokes itself recursively on the two subtrees of T – the subtree rooted at the left child of ρ and the subtree rooted at the right child of ρ. The separators obtained on these two subtrees correspond to separators of $G^{in}_{\rho_\ell}$ and $G^{in}_{\rho_r}$ and their union is returned as the separator for G. In case (2) the algorithm returns the boundary of $\beta(top(z))$ (it could return the boundary of $\beta(top(s))$ instead – it does not make a difference) as the answer. In case (3), we assume without loss of generality that $top(z)$ is the ancestor of $top(s)$, and handling of this case depends on whether z belongs to the boundary of $\beta(top(s))$ or not. In fact, this case splits into three subcases: (3.1) if $z \notin \partial\beta(top(s))$ then the algorithm immediately returns $\partial\beta(top(s))$; (3.2) if $\partial\beta(top(s)) = \{z\}$ then the algorithm invokes itself recursively on $G^{in}_{top(s)}$; and (3.3) if $\partial\beta(top(s)) = \{z, q\}$ for some vertex $q \neq s, z$ then the algorithm first invokes itself recursively on $G^{in}_{top(s)_\ell}$ (assuming $\partial\beta(top(s)_\ell) = \{s, z\}$) and stores the answer in S. If S proves to be a separator in G then S is returned, otherwise, q is added to S and returned. The pseudocode is presented in Algorithm 1.

▶ **Theorem 14.** *Algorithm 1 correctly computes a minimum-sized restricted path (s, z)-temporal separator for a temporal graph G such that $bw(G_\downarrow) \leq 2$.*

Proof. The proof proceeds by the case analysis reflecting the structure of the algorithm. Clearly, the algorithm correctly identifies when there is a separator of size 0 or 1 since it performs brute-force checks for these special cases. Assuming that there is no separator of size ≤ 1, we discuss the correctness for the remaining three cases.

Case (1): $top(s) = top(z) = x \in V(T)$. Observe that Lemma 13, item 1, implies that $s, z \in \partial\beta(x_\ell)$ and $s, z \in \partial\beta(x_r)$. Since the branchwidth is 2, it implies that $\partial\beta(x_\ell) = \partial\beta(x_r) = \{s, z\}$. In addition, we know that $s, z \notin \partial\beta(x)$ by the definition of $top()$. And since every vertex in $\partial\beta(x)$ must appear in $\partial\beta(x_\ell)$ or $\partial\beta(x_r)$ (using Lemma 13, item 2), we conclude that $\partial\beta(x) = \emptyset$. By Lemma 13, item 3, every vertex in G^{in}_x has all its edges from G. Therefore G^{in}_x is disconnected from the rest of G. However, we assume that G is connected, so we must have $G^{in}_x = G$. This is true only when $x = \rho$. Thus, we must have in this case that $top(s) = top(z) = \rho$. Observe that if P is a restricted temporal path between s and z (that does not have s or z as intermediate nodes) then it cannot use edges from both $\beta(\rho_\ell)$ and $\beta(\rho_r)$. Suppose, for contradiction, that P uses both kinds of edges, then there must be a vertex v on this path incident on e_1 and e_2 such that $e_1 \in \beta(x_\ell)$ and $e_2 \in \beta(x_r)$. Since $\beta(x_\ell), \beta(x_r)$ partition all the edges, it implies that $e_2 \notin \beta(x_\ell)$. This means that $v \in \partial\beta(x_\ell) = \{s, z\}$, but $v \neq s, z$, giving a contradiction. Therefore, the minimum size restricted path temporal separator in G is the union of minimum size restricted path temporal separators in $G^{in}_{\rho_\ell}$ and $G^{in}_{\rho_r}$, which is precisely what our algorithm outputs.

Case (2): $top(s)$ and $top(z)$ do not lie on the same root-to-leaf path in T. One of the consequences of Lemma 13, item 3, is that removing $\partial\beta(x)$ from G separates all vertices in $V(G^{in}_x)$ from the rest of the graph. Therefore, removing $\partial\beta(top(z))$ separates all vertices in $G^{in}_{top(z)}$ from the rest of the graph. Observe that $z \in V(G^{in}_{top(z)})$ and $s \notin V(G^{in}_{top(z)})$ (by the condition of this case). Therefore removing $\partial\beta(top(z))$ separates s from z. We claim that this is the minimum separator in this case. This is because when this line is

reached we are guaranteed that there is no separator of size 1, and $|\partial\beta(top(z))| \leq 2$ (in fact, it must be then equal to 2). We only need to be careful that neither z nor s is in $\partial\beta(top(z))$, but it is clear from the definition of $top()$ and the case condition.

Case (3): $top(z)$ is an ancestor of $top(s)$ (if $top(s)$ is an ancestor of $top(z)$ then we can exchange the roles of s and z for the sake of the argument). This case has three subcases.

Subcase (3.1): $z \notin \partial\beta(top(s))$. This is similar to case (2) described above. The algorithm can return $\partial\beta(top(s))$ as a minimum size separator.

Subcase (3.2): $\partial\beta(top(s)) = \{z\}$. In this case, the structure of the graph is such that $G^{in}_{top(s)}$ is connected to the rest of the vertices in G via the node z, while vertex s lies in $G^{in}_{top(s)}$. Thus, to separate z from s, it is sufficient to separate them in $G^{in}_{top(s)}$, which is what the algorithm does.

Subcase (3.3): $\partial\beta(top(s)) = \{z, q\}$. By Lemma 13, item 2, it follows that $s \in \partial\beta(top(s)_\ell)$ and $s \in \partial\beta(top(s)_r)$. By Lemma 13, item 1, it follows that $z, q \in \beta(top(s)_\ell) \cup \beta(top(s)_r)$. Since branchwidth is at most 2, we have (without loss of generality) that $\partial\beta(top(s)_\ell) = \{s, z\}$ and $\partial\beta(top(s)_r) = \{s, q\}$. By an argument similar to the one in case (1), we can establish that any restricted (s, z) temporal path (that does not use s or z as intermediate nodes) must either consist entirely of edges in $\beta(top(s)_\ell)$ or entirely of edges in $\beta(top(s)_r)$. Thus, we can compute the two separators and take their union; however, we can simplify the calculation observing that the only separator we need to consider for the $G^{in}_{top(s)_r}$ is $\{q\}$, since $G^{in}_{top(s)_r}$ is connected to the rest of G only through q and s. ◀

▶ **Corollary 15.** *Given a temporal graph $G = (V, E, \tau)$ with $bw(G_\downarrow) \leq 2$, the problem (s, z, t)-Temporal Separator is solvable in time $O(|V||E||\mathcal{T}|)$ where $\mathcal{T} = \{t(e) : e \in E(s)\}$.*

5.2 Temporal Graphs with a "Tree-like" Underlying Graph

In this section, we present a polynomial time greedy algorithm (motivated by the point-cover interval problem) for computing a path restricted (s, z)-temporal separator (see Section 5.1) on a temporal graph G such that $G_\downarrow \setminus \{s, z\}$ is a tree if the existence of a restricted (s, z)-temporal path could be checked in polynomial time.

We assume that we are given a temporal graph G such that $G_\downarrow \setminus \{s, z\}$ is a tree, which we denote by T. For a pair of nodes (u, w), we let $P_{u,w}$ denote the unique shortest path in T between u and w. For a vertex $v \in V(T)$, we define a removal list of v, denoted by RL_v, to consist of all unordered pairs (u, w) such that $v \in V(P_{u,w})$ and there exists a restricted (s, z)-temporal path in G using the edges of $P_{u,w}$. For a pair $u, w \in V(T)$, we define two temporal graphs: (1) $G^1_{u,w}$ is G induced on the edges of $E(P_{u,w}) \cup \{(s, u), (v, z)\}$, and (2) $G^2_{u,w}$ is G induced on the edges of $E(P_{u,w}) \cup \{(s, v), (u, z)\}$. The removal lists for all vertices in $V(T)$ can be computed efficiently as follows. Initialize all removal lists to be empty. For each pair of vertices $u, w \in V(T)$ check if there is any restricted (s, z)-temporal path in $G^1_{u,w}$ or $G^2_{u,w}$, and if so, then add (u, w) to the removal lists of all nodes in $P_{u,w}$. Let $\mathcal{U} = \bigcup_{v \in V(T)} RL_v$ be the set of all pairs of nodes that appear in removal lists. The following observation is immediate from the definitions and shows that computing a minimum size restricted path (s, z)-temporal separator reduces to covering $\mathcal{U}$ with as few removal lists as possible.

▶ **Observation 16.** *A set of S is a restricted path (s, z)-temporal separator if and only if $\bigcup_{v \in S} RL_v = \mathcal{U}$.*

A vertex v is called topmost if there exists a pair $(u, w) \in RL_v$ such that $(u, w) \notin RL_{parent(v)}$. Our greedy algorithm, called $GreedyRTS$, starts out with an empty solution $S = \emptyset$, and then adds more vertices to S as follows. While there are non-empty removal lists,

the algorithm selects a topmost vertex v with maximum distance from the root of T, adds v to the set S, and removes all pairs in RL_v from the removing lists of all the other vertices. The pseudocode is given in Algorithm 2 (in Appendix B).

For the proof of the next theorem, please see the full version of the paper [17].

▶ **Theorem 17.** *Algorithm 2 computes a minimum-sized restricted path (s, z)-temporal separator in a temporal graph G with $G_{\downarrow} \setminus \{s, z\}$ being a tree.*

Based on Lemma 1, the existence of a (s, z, t)-temporal path can be solved in polynomial time. Thus, the following theorem follows from Theorem 17.

▶ **Theorem 18.** *The (s, z, t)-Temporal Separator problem is solvable in polynomial time on temporal graphs G where $G_{\downarrow} \setminus \{s, z\}$ is a tree.*

5.3 Temporal Graphs with Bounded Pathwidth

In this section, we present a reduction from the *Discrete Segment Covering (DISC-SC)* problem to the (s, z, t)-Temporal Separator problem on graphs with bounded pathwidth. In the DISC-SC problem, we are given a set Γ of n intervals (also called segments), on the rational line and a set $\mathcal{I}$ of unit-intervals on the rational line. We wish to find a subset of unit intervals $A \subseteq \mathcal{I}$ which covers all the segments in Γ. The objective is to minimize the size of A. An interval $I \in \mathcal{I}$ covers a segment $S \in \Gamma$ if at least one endpoint S lies in I. A segment $S \in \Gamma$ is covered by a set of intervals A if there is an interval $I \in A$ that covers S. We refer to the version of DISC-SC where all segments in Γ have length bounded by k as DISC-SC-k. DISC-SC problem is $\mathcal{NP}$-hard [4]. [4] also shows that the DISC-SC problem remains $\mathcal{NP}$-hard when the length of all segments in Γ are equal. DISC-SC-1 can be solved efficiently by a simple greedy algorithm [4]. However, the hardness of DISC-SC-k for general $k > 1$ remains open.

The following theorem serves as a warm-up, and it establishes a simple polynomial time reduction from DISC-SC to the (s, z, t)-Temporal Separator problem. For the proof of the next theorem, please see the full version of the paper [17].

▶ **Theorem 19.** *There is a polynomial-time reduction from the DISC-SC problem to the (s, z, t)-Temporal Separator problem.*

The issue with the above theorem is that it does not provide any structural guarantees about the temporal graph G used in the construction. In order to establish a reduction via a temporal graph G whose underlying graph has bounded pathwidth, we start with a restricted version of DISC-SC, namely, the DISC-SC-k problem. The following results can then be established.

▶ **Theorem 20.** *There is a polynomial-time reduction from the DISC-SC-k problem to the (s, z, t)-Temporal Separator in which the pathwidth of the underlying graph is bounded by $2k + 6$.*

Proof. Consider an instance $(\mathcal{I}, \Gamma)$ of the Discrete Segment Covering problem such that the length of all the segments in Γ is at most k. Consider intervals in $\mathcal{I} = (I_1, I_2, \dots\ I_n)$ in the non-decreasing order of their starting times. We choose a special set of intervals $SP \in \mathcal{I}$ by the following algorithm.

1. Let $SP = I_1$ and $index = 1$.
2. Let j be the largest index such that $s(I_j) < e(I_{index})$, if such j exists. Otherwise, let $j = index + 1$
3. Put I_j into the set SP, update the integer $index$ equal to j and if $j \leq n$ repeat the algorithm from step 2.

For the proof of the next lemma, please see the full version of the paper [17].

▶ **Lemma 21.** *A p is covered by $\mathcal{I}$ if SP covers it.*

The main idea of the proof is based on to the following features of the special set SP. Denote $SP = \{I_{m_1}, I_{m_2}, \dots I_{m_q}\}$. Based on the selection of interval $I_{m_{i+1}}$ it is clear that the starting point of $I_{m_{i+2}}$ is greater than the ending point of I_{m_i} which implies that $s(I_{m_{i+2}}) > s(I_{m_i})+1$. More generally, we have that $e(I_{m_{i+2k}}) > s(I_{m_i}) + k + 1$. Therefore, for any segment $C \in \Gamma$ and for any interval I_{m_i} and I_{m_j} such that $s(C) \in I_{m_i}$ and $e(C) \in I_{m_j}$, we could conclude that $j \leq i + 2k$. This feature for SP is the main idea used in constructing an instance of the (s, z, t)-Temporal Separator problem with low pathwidth.

Now we construct a temporal graph $G = (V, E, \tau)$ where $\tau = |\Gamma| \times t$. Let $V = \{u_i | i \in [n]\} \cup \{v_i | i \in [n]\} \cup \{s, z\}$. Now, for the i-th segment $C \in \Gamma$ we add a path from s to z at time $i \times t$. Let m_a and m_b be the indices of the first intervals in SP which cover points $s(C)$ and $e(C)$, respectively. Based on the Lemma 21 if m_a (or m_b) does not exist, then the point $s(C)$ (respectively, $e(C)$) will not be covered by any interval in $\mathcal{I}$. Therefore, we could treat C as a single point $e(C)$ (respectively, $s(C)$) and continue on with the algorithm. Let l_s be the index of the leftmost interval form $\mathcal{I}$ which covers $s(C)$, and let r_s be the index of the rightmost interval from $\mathcal{I}$ which covers $s(C)$. It is obvious that $s(C)$ is covered by all of the intervals between l_s and r_s in $\mathcal{I}$. Similarly, let l_e and r_e be the indices of the leftmost and the rightmost intervals which cover $e(C)$. If $l_e \leq r_s$ then consider $l_e = r_s + 1$ instead. Now, add the following (s, z, t)-temporal path to the temporal graph G. For simplicity, we denote $i \times t$ by θ.

$$
\begin{aligned}
&(s, u_{l_s}, \theta), (u_{l_s}, v_{l_s}, \theta), (v_{l_s}, v_{l_s+1}, \theta), \dots (v_{r_s-1}, v_{r_s}, \theta)\\
&(v_{r_s}, u_{r_s}, \theta), (u_{r_s}, u_{r_s-1}, \theta), \dots (u_{m_a+1}, u_{m_a}, \theta)\\
&(u_{m_a}, u_{m_b}, \theta)\\
&(u_{m_b}, u_{m_b-1}, \theta), \dots, (u_{l_e+1}, u_{l_e}, \theta), (u_{l_e}, v_{l_e}, \theta)\\
&(v_{l_e}, v_{l_e} + 1, \theta) \dots (v_{r_e-1}, v_{r_e}, \theta), (v_{r_e}, u_{r_e}, \theta), (u_{r_e}, z, \theta)
\end{aligned}
\tag{1}
$$

Figure 2 (in Appendix A) shows the above path in the graph layer $i \times t$. We claim that there exists $A \subseteq \mathcal{I}$ that covers Γ with $|A| \leq p$ if and only if there is a (s, z, t)-temporal separator $S \subseteq V$ such that $|S| \leq p$.

$\rightarrow$ Suppose that $A \subseteq \mathcal{I}$ covers all segments in Γ. Let $S = \{v_i | I_i \in A\}$. It is obvious that $|S| = |A|$. Now we prove that S is a (s, z, t)-temporal separator. Suppose that there is a temporal path P in G, based on the construction of G this temporal path should be of the form shown in equation 1 for some $i \in [n]$. This implies $I_j \notin A$ for all j such that $l_s < j < r_s$ or $l_e < j < r_e$ and results in the i-th segment not being covered by A. So, based on the contradiction we could conclude that S is a (s, z, t)-temporal separator.

$\leftarrow$ Suppose that $S \subseteq V$ is a (s, z, t)-temporal separator in a temporal graph G. Let $A = \{I_i | u_i \in S \text{ or } v_i \in S\}$, it is clear that $|A| \leq |S|$. Consider the i-th segment $C \in \Gamma$. There should be one vertex belonging to the temporal path P which is shown in equation 1 in S since S is a (s, z, t)-temporal separator. Therefore there is j where $l_s < j < r_s$ or $l_e < j < r_e$ and either u_i or v_i belong to S, which implies $C \in A$. Thus, A covers Γ.

Now we prove that the pathwidth of the underlying graph $G_\downarrow = (V, E')$ of the temporal graph $G(V, E, |\Gamma| \times t)$ is bounded by $2k + 6$. We refer to an edge $(u_{m_a}, u_{m_b}, \theta)$ in a path that is shown in equation 1 as a *crossing edge*. Figure 3 (in Appendix A) shows a graph G' of which $G_\downarrow$ is a subgraph. Now we give a path decomposition (P, β) for a graph $G_\downarrow$ in which the width of decomposition is at most $2k + 6$. Let $V(P) = \{a_1, a_2, \dots, a_m\}$ and

$E(P) = \{(a_1, a_2), \ldots, (a_{m-1}, a(m))\}$. Let $i \in [n]$ and $l(i)$ be the largest integer such that the starting point of the interval $I_{m_{l(i)}} \in SP$ is before the starting point of interval I_i. Now we define the $\beta(a_i)$ as follows: $\beta(a_i) = \{u_i, v_i, u_{i+1}, v_{i+1}, s, z\} \cup \{u_{m_l} | l \geq l(i) \text{ and } l \leq l(i) + 2k\}$.

▶ **Lemma 22.** *For any u_q and i, j, l such that $i < j < l$, if $u_q \in \beta(a_i)$ and $u_q \in \beta(a_l)$, we have $u_q \in \beta(a_j)$.*

Proof. If $I_q \notin SP$ then it is clear that u_q only appears in $\beta(a_{q-1})$ and $\beta(a_q)$. Now suppose that $I_1 \in SP$ and $q = m_p$. Since $u_{m_p} \in \beta(a_i)$ we have $m_p \leq l(i) + 2k$, also $l(l) \leq m_p$ since $m_p \in \beta(a_l)$. As a result we have $m_p \leq l(i) + 2k \leq l(j) + 2k$ and $l(j) \leq l(l) \leq m_p$ which implies that $u_q \in \beta(a_j)$. ◀

For any $v_i \in V$ it is clear that v_i just belongs to the two sets $\beta(a_{i-1})$ and $\beta(a_i)$. Also, s and z are present in all the sets. Therefore, by Lemma 1 we could say that the third property of path decomposition is satisfied. So, it is sufficient to show that for every edge $(u, v) \in E(G_\downarrow)$ there exists $i \in [n]$ such that $\{u, v\} \subseteq \beta(a_i)$. If the edges are not crossing edges, then there are three types of edges (u_i, v_i), (u_i, u_{i+1}), and (v_i, v_{i+1}) which satisfy the condition by the definition of $\beta(a_i)$. If $e = (u_i, u_j)$ is a crossing edge, then $I_i \in SP$ and $I_j \in SP$, so let $m_p = i$ and $m_q = j$. Since this edge corresponds to a segment C such that $s(C) \in I_{m_p}$ and $e(C) \in I_{m_q}$ we could conclude that $m_q \leq m_p + 2k$ which implies that $u_i, u_j \subseteq \beta(a_i)$. Also, the cardinality of all sets $\beta(a_i)$ is $2k + 7$ which implies that the width of (P, β) is $2k + 6$. Therefore the pathwidth of the underlying graph $G_\downarrow$ is at most $2k + 6$. ◀

▶ **Theorem 23.** *If the (s, z, t)-Temporal Separator problem on temporal graphs with bounded pathwidth is solvable in polynomial time then the DISC-SC-k problem is solvable in polynomial time.*

6 Conclusions

In this work, we defined the (s, z, t)-Temporal Separator problem, generalizing the (s, z)-Temporal Separator problem. We showed that (s, z)-Temporal Separator and (s, z, t)-Temporal Separator problems could be approximated within τ and τ^2 approximation ratio, respectively, in a graph with lifetime τ. We also presented a lower bound $\Omega(\log(n) + \log(\tau))$ for polynomial time approximability of (s, z, t)-Temporal Separator assuming that $\mathcal{NP} \not\subseteq \text{DTIME}(n^{\log \log n})$. Then we considered special classes of graphs. We presented two efficient algorithms: one for temporal graphs G with $bw(G_\downarrow) \leq 2$ and one for temporal graphs G with $G_\downarrow \setminus \{s, z\}$ being a tree. The question of whether there is a polynomial-time algorithm to compute a minimum (s, z, t)-temporal separator in a temporal graph of bounded treewidth remains an interesting open problem. However, we showed a reduction from the DISC-SC-k problem to (s, z, t)-Temporal Separator when the pathwidth of the underlying graph is bounded by a constant number. Therefore, designing efficient algorithms for bounded treewidth graphs encounters serious obstacles, such as making progress on the open problem of the hardness of DISC-SC-k. Another interesting direction of future research is to consider temporal separator problems with the additional restriction of "balancedness", as discussed at the end of Section 3.

References

1 Susanne Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1-2):3–26, 2003.

2 Haeder Y. Althoby, Mohamed Didi Biha, and André Sesboüé. Exact and heuristic methods for the vertex separator problem. *Computers and Industrial Engineering*, 139:106135, 2020. doi:10.1016/j.cie.2019.106135.

3 Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, Eli Upfal, and Fabio Vandin. Algorithms on evolving graphs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 149–160, 2012.

4 Dan Bergren, Eduard Eiben, Robert Ganian, and Iyad Kanj. On covering segments with unit intervals. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

5 Hans L. Bodlaender and Dimitrios M. Thilikos. Constructive linear time algorithms for branchwidth. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming*, pages 627–637, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

6 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

7 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

8 Xiaojie Deng, Bingkai Lin, and Chihao Zhang. Multi-multiway cut problem on graphs of bounded branch width. In *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, pages 315–324. Springer, 2013.

9 Jessica Enright, Kitty Meeks, George B Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *arXiv preprint*, 2018. arXiv:1805.06836.

10 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

11 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *International Colloquium on Automata, Languages, and Programming*, pages 531–543. Springer, 2004.

12 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Departmental Papers (CIS)*, page 236, 2005.

13 Afonso Ferreira. Building a reference combinatorial model for manets. *IEEE network*, 18(5):24–29, 2004.

14 Paola Flocchini, Bernard Mans, and Nicola Santoro. Exploration of periodically varying graphs. In *International Symposium on Algorithms and Computation*, pages 534–543. Springer, 2009.

15 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. *Theoretical Computer Science*, 806:197–218, 2020.

16 Naveen Garg, Vijay V Vazirani, and Mihalis Yannakakis. Multiway cuts in directed and node weighted graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 487–498. Springer, 1994.

17 Hovhannes A. Harutyunyan, Kamran Koupayi, and Denis Pankratov. Temporal separators with deadlines, 2023. arXiv:2309.14185.

18 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.

19 George B Mertzios, Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. Temporal network optimization subject to connectivity constraints. In *International Colloquium on Automata, Languages, and Programming*, pages 657–668. Springer, 2013.

20 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

21 Neil Robertson and Paul D Seymour. Graph minors. x. obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

22 Neil Robertson and P.D. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

23 Ryan A Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. Modeling dynamic behavior in large evolving graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 667–676, 2013.

24 Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of computer and system sciences*, 26(3):362–391, 1983.

25 Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proceedings of the VLDB Endowment*, 7(9):721–732, 2014.

26 Huanhuan Wu, James Cheng, Yiping Ke, Silu Huang, Yuzhen Huang, and Hejun Wu. Efficient algorithms for temporal path computation. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2927–2942, 2016.

27 B Bui Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.

28 Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *Journal of Computer and System Sciences*, 107:72–92, 2020.

A Figures

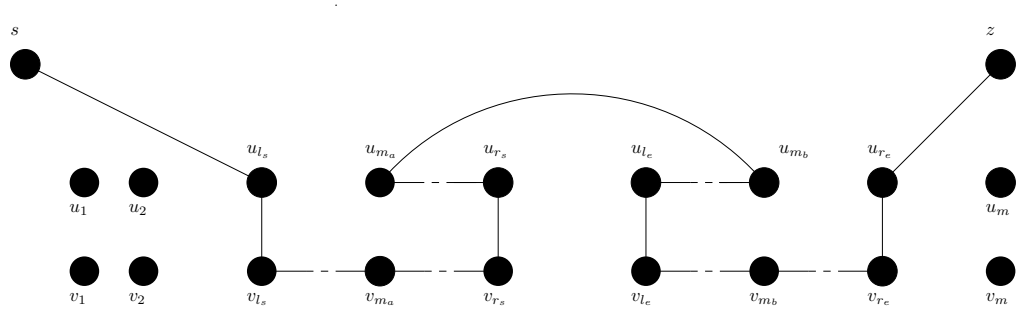


Figure 2 Demonstration of one step of the reduction in the proof of Theorem 20. The figure shows the (s, z, t)-temporal path in the layer $G_{j\times t}$. The time label for all edges is $j \times t$.

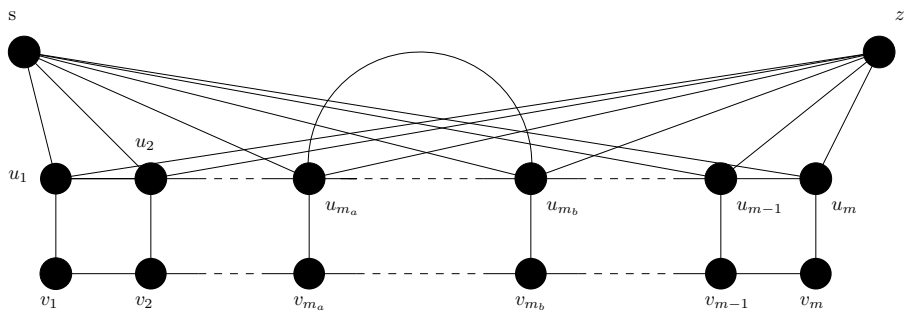


Figure 3 Illustration of the graph G' which is used to show that the output of the reduction from Theorem 20 has bounded pathwidth. The underlying graph $G_\downarrow$ is a subgraph of G'.

B Pseudocode

Algorithm 2 This algorithm computes a minimum sized restricted path (s, z)-temporal separator in a temporal graph G when $G_{\downarrow} \setminus \{s, z\}$ is a tree T.

Function `ComputeRLs`(G, s, z)**:**
  $\mathcal{U} \leftarrow \emptyset$;
  for $(u, w) \in V(T) \times V(T)$ **do**
    if `ExistsRestrictedPath`$(G^1_{u,w}, s, z)$ *or* `ExistsRestrictedPath`$(G^2_{u,w}, s, z)$ **then**
      $\mathcal{U} \leftarrow \mathcal{U} \cup \{(u, w)\}$;
      for $v \in V(P_{u,w})$ **do**
        $RL_v \leftarrow RL_v \cup \{(u, w)\}$;
Function `GreedyRTS`$(G, s, z, RL, \mathcal{U})$**:**
  $S \leftarrow \emptyset$;
  while $\mathcal{U} \neq \emptyset$ **do**
    $v \leftarrow$ furthest node from the root of T such that $\exists (u, w) \in RL_v \setminus RL_{parent(v)}$;
    $S \leftarrow S \cup \{v\}$;
    $\mathcal{U} \leftarrow \mathcal{U} \setminus RL_v$;
    for $w \in V(T)$ **do**
      $RL_w \leftarrow RL_w \setminus RL_v$;
  return S;

Regularization of Low Error PCPs and an Application to MCSP

Shuichi Hirahara
National Institute of Informatics, Tokyo, Japan

Dana Moshkovitz
Department of Computer Science, University of Texas at Austin, TX, USA

Abstract

In a *regular* PCP the verifier queries each proof symbol in the same number of tests. This number is called the *degree* of the proof, and it is at least $1/(sq)$ where s is the soundness error and q is the number of queries. It is incredibly useful to have regularity and reduced degree in PCP. There is an expander-based transformation by Papadimitriou and Yannakakis that transforms any PCP with a constant number of queries and constant soundness error to a regular PCP with constant degree. There are also transformations for low error projection and unique PCPs. Other PCPs are constructed especially to be regular. In this work we show how to regularize and reduce degree of PCPs with a possibly large number of queries and low soundness error.

As an application, we prove NP-hardness of an unweighted variant of the collective minimum monotone satisfying assignment problem, which was introduced by Hirahara (FOCS'22) to prove NP-hardness of MCSP* (the partial function variant of the Minimum Circuit Size Problem) under randomized reductions. We present a simplified proof and sufficient conditions under which MCSP* is NP-hard under the standard notion of reduction: MCSP* is NP-hard under deterministic polynomial-time many-one reductions if there exists a function in E that satisfies certain direct sum properties.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases PCP theorem, regularization, Minimum Circuit Size Problem

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.39

Funding *Shuichi Hirahara*: Supported by JST, PRESTO Grant Number JPMJPR2024, Japan.
Dana Moshkovitz: Supported by the National Science Foundation under grants number 2200956 and 2312573.

Acknowledgements We are thankful to Dean Doron for discussions about explicit construction of dispersers.

1 Introduction

1.1 Regularization of Low Error PCPs

In a Probabilistically Checkable Proof (PCP) the verifier uses randomness to pick a small number of queries to its proof. A correct proof is typically accepted, whereas a proof of an incorrect statement is typically rejected. PCPs found many surprising applications over the years in areas like hardness of approximation [14, 15], cryptography [30], complexity theoretic lower bounds [49, 2, 9], quantum computation [25] and metric embeddings [29].

It is often desirable, both for the construction of PCPs and for their applications, that the PCP is *regular*[1], that is, the verifier queries each proof symbol on the same number of tests. This number is called the *degree*. Some PCPs are naturally regular or can be made regular

[1] We remark that regular PCPs were called *smooth* PCPs in a few forks following the definition of smooth locally decodable codes [27]. In PCP the term "smooth PCP" was also used with a completely different meaning [21]. Hence, we will use the term "regularity" and not "smoothness".

with some effort (see, e.g., [38] that constructed a regular PCP from scratch). In contrast, other constructions are inherently non regular, typically because the proof consists of different parts with different roles. Most constructions, and especially algebraic constructions, do not naturally have small degree. We focus on regularization and degree reduction, i.e., on transformations that take *any* PCP verifier and create a similar PCP verifier that is regular and has small degree.

Note that for a regular PCP with degree d, number of queries q and soundness error s we have $s > 1/(dq)$. The reason is that fraction $1/(dq)$ of the randomness strings have disjoint queries. Since each verifier test is satisfiable on its own, it is always possible to satisfy fraction $1/(dq)$ of the tests. Thus, the degree has to be at least $1/(sq)$. As q is typically constant or only slightly super-constant, one desires d that is about $1/s$.

For soundness error s close to 1 and a small number of queries q, Papadimitriou and Yannakakis [37] showed how to regularize the query graph and make the degree constant. This was extended to the case of low soundness error and unique or projection games ($q = 2$) that is especially important for hardness of approximation [28, 34, 11].

We regularize and decrease degree of PCPs of low soundness error s and a general number q of queries:

▶ **Theorem 1** (Regularization and degree reduction). *Let V be a PCP verifier that uses r random bits to make q queries to a proof over alphabet Σ and has completeness error c and soundness error s, where $s \leq \min\{1/(eq), 1/|\Sigma|^a\}$ for e the basis of the natural logarithm and a constant $0 < a \leq 1$. Then V can be efficiently transformed into a new PCP verifier V' whose query graph is bi-regular and where proof symbols have degree $d = q\mathsf{poly}(1/s)$. The verifier V' uses $r + O(\log(1/s))$ random bits to make $\mathsf{poly}(q)$ queries to a proof over alphabet Σ. It has completeness error c and soundness error $O(s^{\Omega(1)})$.*

We can apply our theorem to the PCPs with the lowest error known today:
For a large constant number of queries:

▶ **Corollary 2** (follows from [10, 12] and Theorem 1). *For any $\beta > 0$, for any $s \geq 2^{-(\log n)^{1-\beta}}$, for every $L \in \mathsf{NP}$, there exists a regular PCP verifier V for L that uses r random bits to make q queries to a proof over alphabet Σ and has perfect completeness and soundness error s, where $r = O(\log n)$, $|\Sigma| = \mathsf{poly}(1/s)$, and $q = \mathsf{poly}(1/\beta)$. The degree is $d = \mathsf{poly}(1/s)$.*

For a super-constant number of queries:

▶ **Corollary 3** (follows from [12] and Theorem 1). *For every $L \in \mathsf{NP}$, there exists a regular PCP verifier V for L that uses r random bits to make q queries to a proof over alphabet Σ and has perfect completeness and soundness error s, where $r = O(\log n)$, $s = 1/\mathsf{poly}(n)$, $|\Sigma| = n^{1/(\log\log n)^{O(1)}}$, and $q = (\log\log n)^{O(1)}$. The degree is $d = \mathsf{poly}(n)$.*

1.2 Regularization Technique

Next we describe the Papadimitriou–Yannakakis transformation and subsequent work, as well as explain the difficulty with a larger number of queries and low soundness error. The idea of Papadimitriou and Yannakakis was to replace each proof symbol with several "copies" of the symbol depending on the symbol's original degree. Whenever the verifier wishes to query the symbol, it queries one of the copies instead, so all copies have low degree. The verifier also checks equality between copies by placing an *expander* of low degree on the set of copies, and picking a random edge from the expander. Overall, the verifier makes an original test with probability $\frac{1}{2}$ and an equality check with probability $\frac{1}{2}$.

This construction is only for the case of large soundness error $s > \frac{1}{2}$, because half of the tests can be satisfied even in the soundness case. Why is this construction only for a small number of queries? Because for an original test *all* q copies queried must be consistent with some global proof π. For the Papadimitrious–Yannakakis verifier this happens if the soundness error is sufficiently larger than $1 - \frac{1}{q}$, so we can tolerate a union bound over the q queries.

To allow for lower soundness error one can combine equality tests with original tests, however the natural way to do it requires $\mathsf{poly}(q, 1/s)$ *queries* in order to ensure that except with probability s the labels to every copy queried are consistent with the majority alphabet symbol for the query. Unfortunately, $1/s$ is large when the soundness error s small. In particular, $1/s$ is often much larger than the number of queries q, so one would get a PCP with a much worse number of queries than the number of queries one started with.

An exception is known for "robust" PCPs[2], for which an increase in the number of queries as described above is acceptable, since robust PCPs can always be converted to PCPs with two queries [11]. Indeed, the transformation outlined above, combining the original test with equality tests is equivalent to the regularization and degree reduction of [34]. Alas, the robust PCPs of lowest error [34, 13] have alphabet size $\exp(1/s)$ instead of $\mathsf{poly}(1/s)$. In particular, to keep the alphabet size polynomial in n for a robust PCP the soundness error has to be at least logarithmically small in n.

We show how to regularize and reduce the degree of general, non robust, PCPs while maintaining $\mathsf{poly}(q)$ queries and $O(s^{\Omega(1)})$ soundness error. The degree becomes $\mathsf{poly}(q, 1/s)$ (recall that degree $1/(qs)$ is needed).

Our construction is similar in spirit to what was described above: we introduce copies for each of the original proof symbols. For each original query the verifier makes queries to several of its copies, checking equality on the copies as well as checking the original test. Surprisingly, we show that only $\mathsf{poly}(q)$ queries chosen according to a disperser suffice. Our main insight is that since the soundness error s of the original PCP is small, it suffices to have a list decoding of size about $\sqrt[q]{1/s}$ for each one of the q queries. There is only probability about $(\sqrt[q]{s})^q = s$ that q copies of an original proof symbol all fall outside the list decoding.

1.3 An Application to MCSP

We expect that our regularization would have many applications in future. Here, we present a specific application of regularization to the Minimum Circuit Size Problem (MCSP) [26].

MCSP is the decision problem that asks to decide whether there exists a circuit of size s that computes a given function $f\colon \{0,1\}^n \to \{0,1\}$, given the truth table of f and a size parameter $s \in \mathbb{N}$. It is easy to see that MCSP $\in \mathsf{NP}$, but it is a long-standing open question whether MCSP is NP-hard. In fact, Levin [32] delayed his publication on the theory of NP-completeness because he hoped to prove NP-hardness of MCSP. In his seminal paper [32], he proved NP-completeness of DNF-MCSP*, i.e, the partial function variant of the Minimum DNF Formula Size Problem. In addition to its historical aspect, MCSP has connections to many areas of theoretical computer science, including learning theory [7], average-case complexity [17], circuit complexity [26, 36, 8], and cryptography [42, 33]. Recently, NP-hardness of the partial function variant of MCSP, denoted by MCSP*, was resolved under randomized polynomial-time reductions [18]. Here, MCSP* is the problem of deciding if there exists a circuit of size s that computes a given *partial* function f on input $x \in f^{-1}(\{0,1\})$, given the truth table of $f\colon \{0,1\}^n \to \{0,1,*\}$ and $s \in \mathbb{N}$ as input.

[2] In a robust PCP [5], in the soundness case, only s fraction of tests are even s-*close* (in Hamming distance) to satisfying. This notion is equivalent to projection PCP [11].

The starting point of the NP-hardness reduction of [18] is the problem called *Collective Minimum Monotone Satisfying Assignment Problem* (CMMSA). The input of CMMSA consists of a collection of monotone formulas of size Δ over n variables, where $\Delta \ll n$, and the task is decide whether there exists an assignment for the n variables with small weight that satisfies as many formulas as possible. In [18], the *weighted* version of CMMSA in which each variable has its own weight is shown to be NP-hard to approximate within a factor of $\Delta^{\Omega(1)}$ using low-error PCP systems. The reason why variables are assigned weights comes from the fact that low-error PCP systems may not be regular. Using our regularization for low-error PCPs, we prove NP-hardness of the unweighted version of CMMSA. This enables us to present a simpler proof of NP-hardness of MCSP*.

Using the simplified proof, we investigate whether MCSP* is NP-hard under the standard notion of reduction. The NP-hardness of MCSP* shown in [18] differs from the standard notion of NP-hardness in that the reduction is *randomized*. The usage of randomized reductions is in some sense necessary because of the connection to a circuit lower bound for explicit functions: A line of work [26, 20, 19, 41] shows that NP-hardness of MCSP* under deterministic reductions implies breakthrough separations, such as $\mathsf{EXP} \neq \mathsf{ZPP}$ or $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$. Thus, proving NP-hardness of MCSP* under deterministic reductions is at least as difficult as the central open problems in complexity theory. In fact, the hardness of certain variants of MCSP under deterministic reductions characterizes some circuit lower bounds for explicit functions [1].

We present sufficient conditions under which the reduction of [18] can be derandomized: MCSP* is NP-hard under deterministic polynomial-time many-one reductions if there exists a family $f = \left\{f_{k,n} \colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}\right\}_{k,n\in\mathbb{N}} \in \mathsf{E} = \mathsf{DTIME}[2^{O(n)}]$ of functions with certain "direct sum" properties. Roughly speaking, for some parameter $\sigma \geq 2^{\Omega(n)}$, we require that (i) $f_{k,n}(i,\text{-})$ can be computed by a circuit of size σ for every $i \in \{0,1\}^k$, and that (ii) for any set $B \subseteq \{0,1\}^k$, the size of any circuit that computes $f_{k,n}(i,\text{-})$ for every $i \in B$ on average is at least $\gtrsim |B| \cdot \sigma$. Although the actual assumption is somewhat stronger, it can be shown that a random function satisfies the direct sum properties with high probability. The original proof of [18] heavily relies on Kolmogorov complexity, and it is unclear what property of random functions is used. Our contribution is to identify the direct sum properties that are sufficient for the proof to go through, by giving a simplified proof that does not rely on Kolmogorov complexity.

We mention that, in general, any randomized polynomial-time reduction for a problem in NP can be derandomized to a deterministic polynomial-time *nonadaptive* reduction that makes several queries, under the assumption that $\mathsf{E} = \mathsf{DTIME}[2^{O(n)}]$ cannot be computed by non-deterministic circuits of size $2^{\Omega(n)}$. This follows from the theory of pseudorandom generators secure against non-deterministic circuits [31, 43]. We also mention that Ilango [22] showed that MCSP* is hard under the Exponential-Time Hypothesis, which provides an *exponential-time* reduction from SAT to MCSP*. Here, we aim at obtaining NP-hardness of MCSP* under deterministic *polynomial-time many-one* reductions.

2 Regularization and Degree Reduction For General PCPs

2.1 Preliminaries

2.1.1 Expanders and Dispersers

We will use an explicit construction of expanders obtained by powering an explicit constant degree expander. For a constant degree expander one can use the construction of [39] based on the zig zag product. The same paper discusses powering as well. The parameters one can get are given in this lemma (for a proof see the appendix of [34]):

▶ **Lemma 4** (Explicit construction of expanders). *There is a constant $\alpha < 1$ and a function $\Delta : \mathbb{N} \to \mathbb{N}^+$ with $\Delta(D) = \Theta(D)$, such that given two natural numbers n and D, one can find in time polynomial in n and in D an undirected (multi-)graph $G = (V, E)$ with $|V| = n$, which is $\Delta(D)$-regular and whose adjacency matrix has second largest eigenvalue (in absolute value) $\lambda \leq (\Delta(D))^\alpha$.*

We will use expander random walk as a hitter (see, e.g., Theorem 4.7 in [47]):

▶ **Lemma 5** (Expander random walk hitting property). *Let $G = (V, E)$ be a Δ-regular undirected (multi-)graph, whose adjacency matrix has second largest eigenvalue (in absolute value) $\lambda\Delta$. Then, for any set $B \subseteq V$ of fraction $\mu = |B| / |V|$, the probability that a random walk $v_1, \ldots, v_t$ in G satisfies $v_i \in B$ for $i = 1, \ldots, t$ is at most $(\mu + \lambda)^t$.*

Lemma 4 and Lemma 5 give an explicit construction of a *disperser*:

▶ **Definition 6** (Disperser/hitter). *A (δ, ε)-disperser graph is a bi-regular bipartite graph $G = (U, V, E)$ such that for every set $B \subseteq V$ of fraction at most ε, for at most δ fraction of the $u \in U$ it holds that all of u's neighbors in G are in B.*

▶ **Corollary 7** (Explicit disperser). *For any $q \geq 1$ and $0 < \varepsilon < 1$, for any $N \geq (1/\varepsilon)^{q+1}$ there is an explicit construction of a $(e\varepsilon^q, \varepsilon)$-disperser $G = ([N], [M], E)$ with N-degree q and M-degree $q \cdot \mathsf{poly}(1/\varepsilon^q)$.*

Proof. Let $G = ([M], E)$ for $M = N/\Delta^q$ be an explicit expander of degree $\Delta = \mathsf{poly}(1/\varepsilon)$ and second eigenvalue $(\varepsilon/q)\Delta$ as given by Lemma 4. Let N correspond to length q walks in G. In the disperser each walk is connected to the q vertices it contains. Let $B \subseteq [M]$ of fraction ε. By Lemma 5, the fraction of walks that fall completely in B is $(\varepsilon+\varepsilon/q)^q = \varepsilon^q(1+1/q)^q \leq e\varepsilon^q$. ◀

2.1.2 PCP Verifiers and Their Parameters

▶ **Definition 8** (PCP verifier). *A PCP verifier for a language L is a procedure that on input x uses r bits of randomness to make q queries to a proof π of length m over alphabet Σ. The verifier satisfies the following:*

- Completeness: *If $x \in L$ then there exists π that the verifier accepts with probability at least c.*
- Soundness: *If $x \notin L$ then for all π the verifier accepts with probability at most s.*

Typically, if n is the input size one considers r that is logarithmic in n, so the answers to all the 2^r tests would make an NP witness if $x \in L$. The number of queries is typically a constant independent of n or slightly super constant; ideally $q = 2$. We have $s \geq 1/|\Sigma|^q$, since a random proof would satisfy at least this fraction of verifier tests. Thus, $|\Sigma|$ is ideally close to $1/s^{1/q}$. The completeness c is often 1. The soundness error s is as small as possible. Sometimes one considers a constant $s < 1$, and sometimes sub-constant s is desired. Note that $s \geq 2^{-r}$. Ideally one could hope for s that is exponentially small in r and polynomially small in n. However, it is currently a known open problem ("The Sliding Scale Conjecture" [4]) whether polynomially small error can be achieved simultaneously with constant number of queries. The state-of-the-art PCP with soundness error $s = 1/n$ has $q = \mathsf{poly}(\log \log n)$ queries [12].

▶ **Definition 9.** *The query graph Q_V of a PCP verifier V that uses r random bits to make q queries to a proof of length m is the bipartite graph that has the 2^r randomness strings of the verifier on one side and the m proof symbols of the proof π on the other side. Connect each randomness string to the q queries the verifier makes on this randomness string.*

2.2 Our Regularization and Its Analysis

Assume a PCP verifier V that uses r random bits to make q queries to a proof π over alphabet Σ and has completeness c and soundness s, where $s \leq 1/(eq), 1/|\Sigma|^a$ for a constant $0 < a \leq 1$ (e is the basis of the natural logarithm). We will construct a new, similar, PCP verifier V' with a bi-regular query graph of small degree as specified in Theorem 1.

Let $A = 1/s$. First, duplicate each of the 2^r tests A times, so each of the degrees is at least A. This causes r to grow to $r + O(\log(1/s))$ and does not change the other parameters. For $l = A, A+1, \ldots, 2^r \cdot A$ consider an explicit disperser $G_l = ([l], [m_l], E_l)$ as guaranteed by Corollary 7 for $N = l$ vertices, $[l]$-degree $q' = \lceil 6q/a \rceil$, and $\varepsilon = s^{1/(2q)}$. Note that $m_l < l$. If V queries the i'th symbol in the proof in $d(i)$ verifier tests, then replace the i'th symbol with $m_{d(i)}$ new symbols $\mathsf{symbol}(i, j)$, $1 \leq j \leq m_{d(i)}$ that are supposed to be copies of the i'th symbol. That is, in the proof for V' in the completeness case all those copies are assigned the same label from Σ as the one assigned to the i'th symbol in the completeness proof of V.

The verifier V' picks a uniformly random test of the verifier V. For every symbol i that the test queries, if the test is the t'th test on which the i'th symbol is queried ($1 \leq t \leq d(i)$), the verifier V' queries instead the q' copies of the i'th symbol that correspond to the $G_{d(i)}$ neighborhood of t. The verifier V' checks equality between the copies in addition to the original test. Overall the number of queries that V' makes is $O(q^2)$, and the degree of every proof symbol is the same $\mathsf{poly}(q, 1/s)$. This step does not change the number of random bits the verifier uses. The alphabet of the proof is the same as the alphabet of the proof of V. The completeness error c of V' is the same as the completeness error c of V.

It remains to prove soundness. Suppose that we have a proof for V' that V' accepts with probability larger than $2\sqrt{s}$.

Let $L = 1/\varepsilon$ and assume for simplicity that L is a natural number (otherwise, round it). For every original proof symbol i, consider the L labels from the alphabet Σ that repeat in the proof of V' in the largest number of copies $\mathsf{symbol}(i, j)$. We call a label from Σ "bad" for i if it is not one of those L. We call a copy "bad" for i if its label is bad for i. Note that a bad label repeats in at most ε fraction of copies.

For any original query i, consider a uniform choice of a V test among the $d(i)$ tests that query it, as well as the q' corresponding queries of V' to copies $\mathsf{symbol}(i, j)$. By the disperser property, for every bad label $\sigma \in \Sigma$ for i, the probability that V' accepts and queries copies labeled σ is at most $e\varepsilon^{q'} \leq es^{3/a} \leq s/(q|\Sigma|)$, where the last inequality used the low soundness error of V. Consider a uniform test of V', which induces a uniform test of V that makes q original queries. By a union bound over the q queries and $|\Sigma|$ possible bad labels for them, the probability that V' accepts yet for one of the q queries it queries a bad copy is at most s.

Hence, with probability larger than $2\sqrt{s} - s \geq \sqrt{s}$ over a choice of a uniform V' test, the verifier accepts and for none of the q original queries it queries a bad copy. Consider the following proof for V: for every proof symbol pick uniformly at random one of its L labels. The probability that this assignment is accepted is larger than $\sqrt{s} \cdot (1/L)^q = s$.

3 Application: NP-Hardness of Partial MCSP

As an application, we simplify the proof of NP-hardness of MCSP* and present two sufficient conditions under which the randomized reductions of [18] can be derandomized. The first sufficient condition is that E cannot be computed by 2^{cn}-time algorithms with $2^n - 2^{n/2}$ bits of advice for a sufficiently large constant c. This condition is essentially equivalent to the statement that there exists a polynomial-time algorithm that, on input 1^N, outputs a string

of length N whose time-bounded Kolmogorov complexity is at least $N - \sqrt{N}$ [40].[3] The second sufficient condition is weaker and is that there exists a function in E that satisfies "direct sum" properties.

To define the direct sum properties formally, we introduce the notion of *oracle-sum circuit*, which generalizes a standard circuit. An oracle-sum circuit consists of a pair (C, D) of an oracle circuit C and a circuit D. The oracle-sum circuit computes a function f such that $f(x) = C^D(x)$, i.e., the function computed by the D-oracle circuit C. Abusing the notation, we identify (C, D) with the function computed by (C, D). The size of an oracle-sum circuit is measured by $|C| + |D|$, where $|C|$ and $|D|$ denote the number of wires in C and D, respectively. Note that it is possible to simulate an oracle-sum circuit (C, D) by a circuit of size $O(|C| \cdot |D|)$ by having $|C|$ copies of the circuit D. The main difference between an oracle-sum circuit and a standard circuit lies in how we measure their size.

For a function $f\colon \{0,1\}^n \to \{0,1\}$, let $f^m\colon (\{0,1\}^n)^m \to \{0,1\}^m$ denote the m-wise direct product of f_i, i.e., the function defined as $f^m(x_1, \dots, x_m) := (f(x_1), \dots, f(x_m))$. We now provide the formal definition of direct sum properties.

▶ **Definition 10.** *For a function $\sigma\colon \mathbb{N}^2 \to \mathbb{N}$, a family $f = \left\{f_{k,n}\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}\right\}_{k,n\in\mathbb{N}}$ of functions is said to have* σ-direct sum properties *if the following hold for some constant $\delta > 0$ and all sufficiently large constant c.*

1. *For every $B \subseteq \{0,1\}^k$, there exists a circuit C of size $|B| \cdot \sigma(k, n)$ such that C computes the m-wise direct product of f_i for every $i \in B$, i.e., $(f_i^m \mid i \in B)$, where $m := n^c$. Here, $f_i(x) := f_{k,n}(i, x)$.*
2. *For every $B \subseteq \{0,1\}^k$ and every oracle-sum circuit C of size at most $|B| \cdot \sigma(k, n) \cdot n^{-1/c}$, there exists $i \in B$ such that $\Pr_{x\sim\{0,1\}^n}[C(i, x) = f_i(x)] \leq 1 - \delta$.*

Roughly speaking, a function with σ-direct sum properties satisfies that the circuit complexity of computing $(f_i \mid i \in B)$ is approximately equal to $\sigma \cdot |B|$ for every $B \subseteq \{0,1\}^k$. Definition 10 is stronger than this in the following respects: (i) Item 1 states that for every $i \in B$, not only each f_i is computable by a circuit of size σ, but also the m-wise direct product of f_i is computable by a circuit of size σ. In particular, computing f_i^m is as easy as computing f_i, which means that the strong direct sum property for f_i *fails* to hold. (ii) Item 2 is a strong direct sum property for $(f_i \mid i \in B)$, and states that *oracle-sum* circuits of size $\lesssim |B| \cdot \sigma$ cannot compute $(f_i \mid i \in B)$ *on average*.

The formal definition of MCSP* is as follows.

▶ **Definition 11.** *For a partial function $f\colon \{0,1\}^n \to \{0,1,*\}$, let $\mathrm{CC}^*(f)$ denote the minimum number of the wires in a circuit C such that $C(x) = f(x)$ for every $x \in \{0,1\}^n$.* Partial MCSP (MCSP*) *is defined as the language that consists of (f, s) such that $\mathrm{CC}^*(f) \leq s$, where f is encoded as a binary string of length $2^{\Theta(n)}$.*

We now state the main result of this section.

▶ **Theorem 12.** *Assume that either*

1. *for every constant $c > 0$, there exists a language in $\mathsf{E} \setminus \mathsf{i.o.DTIME}[2^{cn}]/(2^n - 2^{n/2})$, or*
2. *there exists a family $f = \{f_{k,n}\}_{k\leq n}$ functions computable in time $2^{O(n)}$ with σ-direct sum properties, where $\sigma(k, n) \geq 2^{\gamma n}$ for some constant $\gamma > 0$.*

Then, MCSP* *is* NP*-hard under deterministic polynomial-time many-one reductions.*

We first show that the first condition implies the second condition in Theorem 12. Item 1 of Definition 10 follows from Uhlig's theorem:

[3] We mention that this can be optimized to $N - N^{1-\epsilon}$ for any constant $\epsilon > 0$.

▶ **Lemma 13** ([45, 46]; see also [48]). *Let $r\colon \mathbb{N} \to \mathbb{N}$ be a function such that $r(n) = 2^{o(n/\log n)}$. Then, for all large $n \in \mathbb{N}$, for any function $f\colon \{0,1\}^n \to \{0,1\}$, there exists a circuit of size $O(2^n/n)$ that computes $f^{r(n)}\colon (\{0,1\}^n)^{r(n)} \to \{0,1\}^{r(n)}$.*

▶ **Proposition 14.** *There exists a constant $c > 0$ such that if $f\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}$ cannot be computed by any algorithm running in time 2^{cn} with $2^{n+k} - 2^{n-1}$ bits of advice, then f satisfies the σ-direct sum property for $\sigma = \Theta(2^n/n)$.*

Proof. To see the first property of Definition 10, for each $i \in B$, by Lemma 13, there exists a circuit C_i of size $O(2^n/n)$ that computes the m-wise direct product f_i^m of f_i. By combining the circuits $(C_i \mid i \in B)$ for all $i \in B$, we obtain a circuit of size $|B| \cdot O(2^n/n)$ that computes f_i^m for every $i \in B$.

Let $\mathrm{K}^t(x)$ denote the t-time bounded Kolmogorov complexity of x, i.e., the length of a shortest program that prints x in time t. Then, the assumption implies that $\mathrm{K}^{2^{cn}}(f) \geq 2^{n+k} - 2^{n-1}$.

To see the second property, we first claim that for every $B \subseteq [n]$, the time-bounded Kolmogorov complexity of $f_B := (f_i \mid i \in B)$ is at least $|B| \cdot 2^{n-2}$. Since f can be described by a description for $(f_i \mid i \in B)$, the set B, and $(f_i \mid i \in \{0,1\}^k \setminus B)$, we have

$$2^{n+k} - 2^{n-1} \leq \mathrm{K}^t(f) \leq \mathrm{K}^{t'}(f_B) + |B| \cdot O(k) + (2^k - |B|) \cdot 2^n$$

for some $t, t' \leq 2^{O(n)}$. It follows that $\mathrm{K}^t(f_B) \geq |B| \cdot (2^n - 2^{n-1} - O(k)) \geq |B| \cdot 2^{n-2}$. We now claim that any small oracle-sum circuit fails to approximate $(f_i \mid i \in B)$. Let C be an oracle-sum circuit of size s such that $\Pr_{x \sim \{0,1\}^n}[C(i,x) = f_i(x)] \geq 1 - \delta$ for every $i \in B$. For each $i \in B$, the set of inputs x such that $C(i,x) \neq f_i(x)$ can be described by $\log \sum_{k \leq \delta 2^n} \binom{2^n}{k} \leq \mathrm{H}_2(\delta) 2^n \leq 2^{n-3}$ bits, where the last inequality holds for a sufficiently small constant $\delta > 0$. Since C can be described by $O(s \log s)$ bits, $(f_i \mid i \in B)$ can be described by $O(s \log s) + |B| \cdot 2^{n-3}$. Thus, we obtain $|B| \cdot 2^{n-3} \leq O(s \log s)$, which implies $s \geq \Omega(|B| \cdot 2^n/n)$. ◀

Since a random function has high Kolmogorov complexity, the proof of Proposition 14 also shows that a random function satisfies $\Theta(2^n/n)$-direct sum properties with high probability.

3.1 Collective Minimum Monotone Satisfying Assignment Problem

In [18], *Collective Minimum Monotone Satisfying Assignment* (CMMSA) was introduced and shown to be NP-hard to approximate. Using the regularization for low-error PCPs, we show that the same hardness of approximation can be proved for the unweighted version of CMMSA.

For an assignment $\alpha\colon [n] \to \{0,1\}$, let $w(\alpha)$ denote the Hamming weight $\sum_{i=1}^n \alpha(i)$ of α. For a formula φ, let $\varphi(\alpha) \in \{0,1\}$ denote the output of φ when the variables are assigned by α.

▶ **Definition 15** ([18]). *The* Collective Minimum Satisfying Assignment problem (CMMSA) with gap $g \in \mathbb{N}$ and soundness $\epsilon > 0$ *is the following problem. The input consists of a collection $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ of monotone formulas over the set $[n]$ of variables and a threshold parameter $s \in \mathbb{N}$. The task is to distinguish the following two cases.*

Yes: *There exists an assignment $\alpha\colon [n] \to \{0,1\}$ such that*

$$w(\alpha) \leq s \quad \text{and} \quad \Pr_{\varphi \sim \Phi}[\varphi(\alpha) = 1] = 1.$$

No: *For every assignment* $\alpha\colon [n] \to \{0,1\}$*, if* $w(\alpha) \le g \cdot s$*, then*

$$\Pr_{\varphi\sim\Phi}[\varphi(\alpha) = 1] < \epsilon.$$

The degree *of* Φ *is defined to be* $\max_{\varphi\in\Phi} |\varphi|$*, where* $|\varphi|$ *denotes the number of the literals in the formula* φ*. The size of an instance of* CMMSA *is measured by the number* n *of input variables.*

▶ **Theorem 16.** *For any constant* $\beta > 0$*, there exists a constant* $\alpha > 0$ *such that for every parameter* $\Delta\colon \mathbb{N} \to \mathbb{N}$ *such that* $\omega(1) \le \Delta(n) \le 2^{(\log n)^{1-\beta}}$ *for all large* $n \in \mathbb{N}$*, it is* NP*-hard under polynomial-time many-one reductions to compute* CMMSA *with gap* $\Delta(n)^{\alpha}$*, degree* $\Delta(n)$*, and soundness* $\Delta(n)^{-\alpha}$ *on a collection* Φ *of monotone DNF formulas over* n *variables.*

The proof of Theorem 16 is essentially the same with [18], except that we use the regularized PCP system of Corollary 2.

Proof of Theorem 16. The PCP theorem of Corollary 2 can be stated in terms of MaxCSP as follows: Let $\Psi = \{C_1, \dots, C_m\}$ be the set of constraints over n variables on the alphabet Σ. Here, for any internal randomness $j \in \{0,1\}^{O(\log n)}$ of a PCP verifier, there is a constraint C_j. Each constraint C_j depends on $D = O(1/\beta)$ variables. The size of the alphabet Σ is at most $\mathsf{poly}(1/\delta)$, where δ is the soundness error. Let $C_j^{-1}(1)$ denote the set of assignments to the variables in C_j that cause C_j to accept. Here, an assignment r to the variables in C_j is a function $r\colon \mathrm{dom}(r) \to \Sigma$, where $\mathrm{dom}(r) \subseteq [n]$ denotes the set of variables in C_j.

Given the MaxCSP instance Ψ over Σ, we reduce it to an instance (Φ, s) of CMMSA as follows: Each variable of Φ is indexed by $(x, a) \in [n] \times \Sigma$ and is denoted by $L_{x,a}$. Informally, $L_{x,a} = 1$ indicates that the variable x in the original CSP instance Ψ is assigned to $a \in \Sigma$. For each $j \in [m]$, construct a monotone DNF formula φ_j defined as

$$\varphi_j(L) := \bigvee_{r\in C_j^{-1}(1)} \bigwedge_{x\in\mathrm{dom}(r)} L_{x,r(x)}.$$

The threshold s is defined to be n.

We prove the correctness of the reduction. Assume that the CSP instance Ψ is satisfied by an assignment $\alpha\colon [n] \to \Sigma$. Then, we set $L_{x,\alpha(x)} := 1$ and $L_{x,y} := 0$ for every $y \in \Sigma \setminus \{\alpha(x)\}$. The weight of the assignment $L\colon [n] \times \Sigma \to \{0,1\}$ is $w(L) = n$. By the perfect completeness, we have $C_j(\alpha) = 1$ for every $j \in [m]$; thus, α satisfies every formula in Φ. It follows that (Φ, s) is a Yes instance of CMMSA.

Next, assume that any assignment to Ψ can satisfy at most a δ-fraction of constraints in Ψ. Assume that there exists an assignment $L\colon [n] \times \Sigma \to \{0,1\}$ such that $w(L) = g \cdot n$ and

$$\Pr_{j\sim[m]}[\varphi_j(L) = 1] \ge \epsilon, \tag{1}$$

where $\epsilon > 0$ is a parameter to be chosen later. We claim that g must be large. For each variable $x \in [n]$ of Ψ, let $A(x) := \{a \in \Sigma \mid L_{x,a} = 1\}$. Since $gn = w(L) = \sum_{x\in[n]} |A(x)|$, we have $\mathbf{E}_{x\sim[n]}[|A(x)|] = g$. We say that $x \in [n]$ is *bad* if $|A(x)| \ge 2gD/\epsilon$. By Markov's inequality, the probability that x is bad is at most $\epsilon/2D$. Since the PCP system is bi-regular, the uniform distribution $x \sim [n]$ is identical to the following distribution: First choose $j \sim [m]$, and then choose x uniformly at random from the variables in C_j. We say that C_j is bad if C_j contains some bad variable x. Thus, the probability, over $j \sim [m]$, that C_j is bad is at most $\epsilon/2$. Combining this with Equation (1), we obtain that

$$\Pr_{C\sim\Psi}\left[\exists r \in C^{-1}(1), \forall x \in \mathrm{dom}(r),\ |A(x)| \le \frac{2gD}{\epsilon} \text{ and } r(x) \in A(x)\right] \ge \epsilon - \frac{\epsilon}{2} = \frac{\epsilon}{2}.$$

Now, we construct a random assignment $\alpha\colon [n] \to \Sigma$ as follows: For each $x \in [n]$, pick $a \sim A(x) \subseteq \Sigma$ uniformly and randomly and define $\alpha(x) := a$. Under the event that $r \in C^{-1}(1)$, $|A(x)| \leq \frac{2gD}{\epsilon}$, and $r(x) \in A(x)$ for every $x \in \mathrm{dom}(r)$, we have $C(\alpha) = 1$ if $\alpha(x) = r(x)$ for every $x \in \mathrm{dom}(r)$, which happens with probability at least $\left(\frac{\epsilon}{2gD}\right)^D$. It follows that

$$\delta \geq \Pr_{\substack{C \sim \Psi \\ \alpha}}[C(\alpha) = 1] \geq \frac{\epsilon}{2} \cdot \left(\frac{\epsilon}{2gD}\right)^D,$$

which implies that $g \geq \Omega\left(\epsilon^2 \cdot \delta^{-\frac{1}{D}}\right) \geq \Omega\left(\delta^{-\frac{1}{2D}}\right)$, where the last inequality holds by setting $\epsilon := \delta^{\frac{1}{4D}}$. The number of the literals in $\varphi_j \in \Phi$ is at most $|C_j^{-1}(1)| \cdot D \leq |\Sigma|^D \cdot D \leq \delta^{-O(D)}$. Given a parameter Δ, we choose $\delta := \Delta^{-\Omega(1/D)}$ so that the degree of Φ is at most Δ. Then, the gap g is at least $\Omega\left(\delta^{-\frac{1}{2D}}\right) \geq \Delta^{\Omega(1/D^2)}$. Moreover, the soundness ϵ is at least $\delta^{\frac{1}{4D}} \geq \Delta^{-\Omega(1/D^2)}$. ◀

3.2 Technical Tools

We review the three technical tools used in [18]. The first tool is a secret sharing scheme.

▶ **Definition 17** (Secret Sharing Scheme [3]). *A* secret sharing scheme *for $\mathcal{A} \subseteq 2^{[n]}$ is a pair* $(\mathrm{Share}, \mathrm{Rec})$ *of a randomized algorithm* Share *and a deterministic algorithm* Rec *with the following properties:*

Correctness: *For every authorized set $T \in \mathcal{A}$ and for every bit $b \in \{0,1\}$, the output of* $\mathrm{Share}(b)$ *is a sequence $(s_1, \ldots, s_n)$ of n strings that satisfies* $\mathrm{Rec}(T, s_T) = b$ *with probability* 1 *over the internal randomness of* $\mathrm{Share}(b)$.

Privacy: *For every unauthorized set $T \notin \mathcal{A}$ and for every random variable b on $\{0,1\}$, the random variables b and $\mathrm{Share}(b)_T$ are statistically independent.*

For a monotone formula φ, let $\mathcal{A}_\varphi$ denote the access structure such that $T \in \mathcal{A}_\varphi$ if and only if the indicator function of $T \subseteq [n]$ satisfies φ.

▶ **Lemma 18** ([24, 6]). *Let $\mathcal{A} = \{\mathcal{A}_\varphi\}_\varphi$ be the family of access structures $\mathcal{A}_\varphi$ represented by monotone formulas φ. Then, there exists a pair of a randomized polynomial-time algorithm* Share *and a deterministic polynomial-time algorithm* Rec *such that for every monotone formula φ, the pair* $(\mathrm{Share}(\varphi, \text{-}), \mathrm{Rec}(\varphi, \text{-}))$ *is a secret sharing scheme for the access structure $\mathcal{A}_\varphi$. Moreover, the length $|s_i|$ of each share s_i is at most the number $|\varphi|$ of the literals in the formula φ.*

The second tool is the Nisan–Wigderson pseudorandom generator construction.

▶ **Proposition 19** ([35, 44]). *For any sufficiently large parameters $\ell, m, \rho \in \mathbb{N}$ with $m \leq 2^\ell$, there exists a "design" $S_1, \ldots, S_m \subseteq [d]$ such that for every $i \in [m]$,*

1. *$|S_i| = \ell$, $d = O(\exp(\ell/\rho) \cdot \ell^2/\rho)$, and*
2. *$|S_i \cap S_j| \leq \rho$ for every $j \in [m] \setminus \{i\}$*

Moreover, such a family can be constructed in time $\mathsf{poly}(2^d, m)$.

▶ **Definition 20** (The Nisan–Wigderson pseudorandom generator construction [35]). *Let $\mathcal{S} = (S_1, \ldots, S_m)$ be a family of ℓ-sized subsets of $[d]$. For a function $f\colon \{0,1\}^\ell \to \{0,1\}$, we define a function*

$$\mathrm{NW}_{\mathcal{S}}\colon \{0,1\}^{2^\ell} \times \{0,1\}^d \to \{0,1\}^m$$

as

$$\mathrm{NW}_{\mathcal{S}}(f;z) := (f(z_{S_1}),\dots,f(z_{S_m})) \in \{0,1\}^m,$$

where $z_{S_i} \in \{0,1\}^\ell$ *denotes the string obtained by concatenating all the bits of* $z \in \{0,1\}^d$ *indexed by* $S_i \subseteq [d]$.

The third tool is a derandomized version of Yao's XOR lemma.

▶ **Lemma 21** ([23, 16]; see also [18]). *For any constant* $\gamma > 0$*, there exist a constant* $c \in \mathbb{N}$ *and a procedure* Amp *that takes a function* $f\colon \{0,1\}^n \to \{0,1\}$ *and parameters* $\epsilon, \delta \in (0,1/2)$ *as input, and returns a function* $\mathrm{Amp}^f = \mathrm{Amp}^f_{\epsilon,\delta}\colon \{0,1\}^{cn} \to \{0,1\}$ *that satisfies the following properties:*

1. *For every circuit* D *that computes* Amp^f *on a* $(1/2+\epsilon)$*-fraction of inputs, there exists an oracle circuit* C *of size* $2^{\gamma n} \cdot \mathsf{poly}(1/\epsilon\delta)$ *such that* C^D *computes* f *on a* $(1-\delta)$*-fraction of inputs.*
2. *There is a nonadaptive* f*-oracle circuit of size* $\mathsf{poly}(n/\epsilon\delta)$ *and depth* $O(\log(n/\epsilon\delta))$ *that computes* Amp^f *by making* $O(1/\epsilon\delta)$ *queries to* f*.*
3. Amp^f *can be computed in time* $\mathsf{poly}(2^n, n/\epsilon\delta)$ *given the truth table of* f *and the parameters as input.*

3.3 Proof of NP-hardness of MCSP*

We are ready to present a proof of Theorem 12. As shown in Proposition 14, the first condition is stronger than the second condition. Thus, it suffices to show NP-hardness of MCSP* under the second condition that there exists a family $F = \{F_{k,n}\}_{k\le n}$ of functions with σ-direct sum properties, where $\sigma(k,n) \ge 2^{\gamma n}$. Let $\delta > 0$ be the constant of Definition 10.

We present a reduction from CMMSA with degree Δ and soundness ϵ_0 to MCSP*, where $\Delta := (\log n)^{1/2}$ and $\epsilon_0 < 1/4$. Let (Φ,θ) be an instance of CMMSA, where $\Phi = \{\varphi_1,\dots,\varphi_\nu\}$ is a degree-Δ collection of monotone formulas over the set $[n]$ of input variables. For each $j \in [\nu]$, let V_j denote the set $\left\{v^j_1,\cdots,v^j_m\right\} \subseteq [n]$ of the variables of φ_j. Here, $m \le \Delta$ is the number of variables on which φ_j depends for every $j \in [\nu]$. We may assume without loss of generality that m does not depend on j and n is a power of 2.

Let Amp be the hardness amplification procedure of Lemma 21 for $\epsilon := \epsilon_0/2\Delta m$, and let $c \ge 1$ be the constant of Lemma 21. Let $\lambda = n^{O(1)}$ be a sufficiently large parameter. We define $\ell := c\log\lambda$.

Let $f := F_{\log n,\log\lambda}\colon \{0,1\}^{\log n} \times \{0,1\}^{\log\lambda} \to \{0,1\}$, where we identify $\{0,1\}^{\log n}$ with $[n]$. For each $k \in [n]$, let $f_k\colon \{0,1\}^{\log\lambda} \to \{0,1\}$ be the function such that $f_k(x) = f(k,x)$ for every $x \in \{0,1\}^{\log\lambda}$. Let $\widehat{f_k} := \mathrm{Amp}^{f_k}\colon \{0,1\}^\ell \to \{0,1\}$ denote the hardness-amplified version of the function f_k.

We construct a partial function

$$g\colon \{0,1\}^{O(\log\nu)} \times \{0,1\}^d \times (\{0,1\}^\Delta)^m \to \{0,1,*\},$$

which is the output of the reduction, as follows. Let $\mathcal{S} = (S_1,\dots,S_{m\Delta})$ be the collection of ℓ-sized subsets of $[d]$ from Proposition 19, where $d = O(\ell)$ and $\rho := \gamma\log\lambda$. Let $\mathcal{S}_i := (S_{(i-1)\Delta+1},\dots,S_{(i-1)\Delta+\Delta})$ for every $i \in [m]$. g takes $x = (j,z,\xi_1,\dots,\xi_m) \in \{0,1\}^{O(\log\nu)} \times \{0,1\}^d \times (\{0,1\}^\Delta)^m$ as input. Define $s_i := \xi_i \oplus \mathrm{NW}_{\mathcal{S}_i}\left(\widehat{f_{v^j_i}};z\right)$ for every $i \in [m]$, where $\oplus$ denotes the bit-wise XOR. Then, we check if $(s_1,\dots,s_m)$ can be obtained by running $\mathrm{Share}(\varphi_j,b)$ for some secret $b \in \{0,1\}$ and some internal randomness of Share. (Here, Share

is the randomized algorithm of Lemma 18.) If not, we define $g(x) := *$; otherwise, we define $g(x) := b$. Observe that $|x| = O(\log \nu + d + \Delta m) = O(\log n + \Delta^2) = O(\log n)$.

We prove the correctness of the reduction that maps (Φ, θ) to an instance $(g, 2\theta\sigma)$ of MCSP* in the following two claims.

▷ Claim 22. If (Φ, θ) is a Yes instance of CMMSA, then $\mathrm{CC}^*(g) \leq \theta\sigma + \mathsf{poly}(n \log \lambda/\epsilon\delta)$, where $\sigma := \sigma(\log n, \log \lambda)$.

Since $\mathsf{poly}(n \log \lambda/\epsilon\delta) \leq 2^{\gamma \log \lambda} \leq \sigma$ for a sufficiently large λ, it follows that $\mathrm{CC}^*(g) \leq 2\theta\sigma$ in the Yes case.

Proof of Claim 22. Let $\alpha\colon [n] \to \{0,1\}$ be an assignment of weight θ that satisfies every formula in Φ. Define $T := \alpha^{-1}(1)$. We construct a circuit C that computes the partial function g. Let $x = (j, z, \xi_1, \ldots, \xi_m)$ be an input to C. First, for each $k \in T$, the circuit C computes strings $(y_1^k, \ldots, y_\Delta^k) \in \left(\{0,1\}^\ell\right)^\Delta$ such that if there exists $i \in [m]$ such that $k = v_i^j$, then $y_p^k = z_{S_{(i-1)\Delta+p}}$ for every $p \in [\Delta]$. Then, for each $k \in T$, the circuit C computes $(\widehat{f_k}(y_1^k), \ldots, \widehat{f_k}(y_\Delta^k))$ from $(y_1^k, \ldots, y_\Delta^k)$. By Lemma 21, each $\widehat{f_k}(y_p^k)$ can be computed by $O(1/\epsilon\delta)$ nonadaptive queries to f; thus, the tuple $\left(\widehat{f_k}(y_p^k) \;\middle|\; p \in [\Delta]\right)$ can be computed by $O(\Delta/\epsilon\delta)$ nonadaptive queries to f_k. By the direct sum property of F, the nonadaptive queries to f_k for every $k \in T$ can be simulated by a circuit of size $|T| \cdot \sigma \leq \theta \cdot \sigma$. Finally, C computes $s_i := (\widehat{f_k}(y_1^k), \ldots, \widehat{f_k}(y_\Delta^k)) \oplus \xi_i$ for every k and i such that $k = v_i^j$. Then, C outputs $b = \mathrm{Rec}(\varphi_j, V_j \cap T, s_{V_j \cap T})$. Overall, the size of the circuit is

$$\theta \cdot \sigma + \mathsf{poly}(n\Delta \log \lambda/\epsilon\delta). \qquad \lhd$$

Let $\eta > 0$ be a constant such that CMMSA is NP-hard to approximate to within a factor of $4(\log \lambda)^\eta$. For a No instance (Φ, θ) of CMMSA, we claim that $\mathrm{CC}^*(g)$ is large.

▷ Claim 23. Assume that

$$\Pr_{j \sim [\nu]}[\varphi_j(\alpha) = 1] \leq \epsilon_0$$

for every assignment α of weight $4\theta \cdot (\log \lambda)^\eta$. Then,

$$\mathrm{CC}^*(g) > 2\theta\sigma.$$

Proof of Claim 23. Let C be an arbitrary circuit of size $2\theta\sigma$. We prove that C cannot compute g on average with respect to some distribution.

We say that C *knows* $k \in [n]$ if there exists an oracle circuit S of size t such that S^C computes f_k on a $(1-\delta)$-fraction of inputs, where $t := 2^{\gamma \log \lambda} \cdot \mathsf{poly}(m\Delta/\epsilon\delta)$. Let B be the set of $k \in [n]$ such that C knows k.

For every $j \in [\nu]$ and every $i \in \{0, \ldots, m\}$, we consider the hybrid distribution H_i^j defined by the following sampling procedure: Choose a secret $b \sim \{0,1\}$ randomly. Let $(s_1, \ldots, s_m) := \mathrm{Share}(\varphi_j, b)$. Define

$$x := \left(j, z, Y_1, \ldots, Y_i, Y'_{i+1}, \ldots, Y'_m\right),$$

where $Y_a := \mathrm{NW}_{\mathcal{S}_a}\left(\widehat{f_{v_a^j}}; z\right) \oplus s_a$ for every $a \in [m]$ and $Y'_a := Y_a$ if $v_a^j \in B$ and $Y'_a \sim \{0,1\}^\Delta$ otherwise. Output (x, b).

Fix any $j \in [\nu]$ and $i \in [m]$. We claim that

$$\left| \Pr_{(x,b) \sim H_{i-1}^j}[C(x) = b] - \Pr_{(x,b) \sim H_i^j}[C(x) = b] \right| \leq \epsilon\Delta. \tag{2}$$

Assume, towards a contradiction, that this does not hold. The only difference between H^j_{i-1} and H^j_i is that the i-th coordinate is Y'_i in the former and is Y_i in the latter. Let $k := v^j_i$. If $k \in B$, it is evident that the two distributions are identical, in which case we are done. Thus, assume $k \notin B$. In this case, Y'_i is the uniform distribution. We use a standard security proof of the Nisan–Wigderson generator to construct a C-oracle circuit S^C of size t that computes f_k, which contradicts $k \notin B$. Specifically, we use a hybrid argument in which each bit of Y_i is replaced with Y'_i. Then, there exists a bit position a of Y_i that can be distinguished from the uniform distribution. We fix $z_{[m\Delta]\setminus S_a}$, b, the randomness of $\mathrm{Share}(\varphi_j, b)$ so that the distinguishing probability is preserved. Since $|z_{S_a} \cap z_{S_{a'}}| \le \rho$ for every $a' \neq a$, given z_{S_a}, one can compute $\left(\widehat{f_k}(z_{S_{a'}}) \;\middle|\; a' \in [m\Delta] \setminus \{a\}\right)$ using a circuit of size $O(m\Delta 2^\rho \rho) \le \lambda^\gamma \cdot \mathsf{poly}(m\Delta)$. By Yao's next bit predictor, we obtain a C-oracle circuit that computes $\widehat{f_k}(z_{S_a})$ on a $(1/2+\epsilon)$-fraction of inputs. By Lemma 21, we obtain a C-oracle circuit S^C that computes f_k on a $(1-\delta)$-fraction of inputs. The size of S is at most $\lambda^\gamma \cdot \mathsf{poly}(m\Delta/\epsilon\delta) \le t$, which implies $k \in B$. However, this contradicts $k \notin B$.

It follows from Equation (2) that

$$\left| \Pr_{(x,b)\sim H^j_0}[C(x) = b] - \Pr_{(x,b)\sim H^j_m}[C(x) = b] \right| \le \epsilon \Delta m.$$

Observe that $g(x) = b$ for every (x, b) in the support of H^j_m. Thus, we have

$$\Pr_{(x,b)\sim H^j_m}[C(x) = b] = \Pr_{(x,b)\sim H^j_m}[C(x) = g(x)]$$

Let $\alpha\colon [n] \to \{0,1\}$ be a function such that $\alpha(k) = 1$ iff $k \in B$. In the distribution of $(x, b) \sim H^j_0$, only the shares in B are included in x. Thus, if $\varphi_j(\alpha) = 0$, then by the privacy of the secret sharing scheme, b and x are statistically independent, in which case we have

$$\Pr_{(x,b)\sim H^j_0}[C(x) = b] = \frac{1}{2}.$$

We claim that the size of B is small. In order to use the direct sum property of F, we construct a small oracle-sum circuit (S, C) that approximates f_k for every $k \in B$. For each $k \in B$, let S_k be the oracle circuit S_k of size t such that S^C_k computes f_k on a $(1-\delta)$-fraction of inputs. We define an oracle circuit S^C as follows: Given $k \in B$ and $x \in \{0,1\}^n$ as input, the circuit outputs $S^C_k(x)$. The size of S is at most $(1 + o(1)) \cdot |B| \cdot t$. The size of the oracle-sum circuit (C, S) is at most $|S| + |C|$. By the direct sum property of F, we obtain $|B| \cdot \sigma \cdot (\log \lambda)^{-\eta} \le |S| + |C| \le (1 + o(1)) \cdot |B| \cdot t + |C|$. Since $t \ll \sigma \cdot (\log \lambda)^{-\eta}$, we obtain $|B| \le (1 + o(1)) \cdot |C| \cdot (\log \lambda)^\eta / \sigma \le \theta \cdot 4(\log \lambda)^\eta$. By the assumption, we have $\Pr_j[\varphi_j(\alpha) = 1] \le \epsilon_0$.

Choose $j \sim [\nu]$ randomly. Then, we obtain

$$\begin{aligned}\Pr_{j\sim[\nu],(x,b)\sim H^j_m}[C(x) = g(x)] &\le \Pr_j[\varphi_j(\alpha) = 1] + \Pr_{j,(x,b)}[C(x) = g(x) \mid \varphi_j(\alpha) = 0] \\ &\le \epsilon_0 + \frac{1}{2} + \epsilon\Delta m \le \frac{1}{2} + 2\epsilon_0 < 1.\end{aligned}$$

◁

4 Open Problems

Can we show NP-hardness of MCSP* under circuit lower bound assumptions, such as the assumption that E cannot be computed by non-deterministic circuits of size $2^{\epsilon n}$ for some constant $\epsilon > 0$? Using a pseudorandom generator secure against non-deterministic

algorithms [31, 43], one can generate, in time $\mathsf{poly}(N)$, functions $f_1, \ldots, f_N \colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}$ such that f_i satisfies the $\Theta(2^n/n)$-direct sum properties for most $i \in [N]$, where $N = 2^{O(n+k)}$. However, it remains open whether a *single* function f with direct sum properties can be obtained.

It is interesting to see whether there exists any candidate for a function with σ-direct sum properties, where $2^{\Omega(n)} \leq \sigma \ll 2^n/n$. In this regime, Uhlig's theorem (Lemma 13) cannot be used, so new insights would be required to answer this question.

The original motivation of the regularization was to show NP-hardness of learning sparse parities by small programs, which was raised as an open problem in [18]. Unfortunately, it turned out that regularization is not sufficient for resolving this question. It remains open whether learning sparse parities by small programs is NP-hard.

References

1 E. Allender and S. Hirahara. New Insights on the (Non-)Hardness of Circuit Minimization and Related Problems. *TOCT*, 11(4):27:1–27:27, 2019. doi:10.1145/3349616.

2 J. Alman and L. Chen. Efficient construction of rigid matrices using an NP oracle. In *Proc. 60th IEEE Symp. on Foundations of Computer Science*, pages 1034–1055, 2019.

3 A. Beimel. Secret-Sharing Schemes: A Survey. In *The Third International Workshop on Coding and Cryptology (IWCC)*, pages 11–46, 2011. doi:10.1007/978-3-642-20901-7_2.

4 M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximations. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 294–304, 1993.

5 E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006.

6 J. C. Benaloh and J. Leichter. Generalized Secret Sharing and Monotone Functions. In *Proceedings of the International Cryptology Conference (CRYPTO)*, pages 27–35, 1988. doi:10.1007/0-387-34799-2_3.

7 M. L. Carmosino, R. Impagliazzo, V. Kabanets, and A. Kolokolova. Learning Algorithms from Natural Proofs. In *Proceedings of the Conference on Computational Complexity (CCC)*, pages 10:1–10:24, 2016. doi:10.4230/LIPIcs.CCC.2016.10.

8 L. Chen, S. Hirahara, I. C. Oliveira, J. Pich, N. Rajgopal, and R. Santhanam. Beyond Natural Proofs: Hardness Magnification and Locality. In *Proceedings of the Innovations in Theoretical Computer Science Conference (ITCS)*, pages 70:1–70:48, 2020. doi:10.4230/LIPIcs.ITCS.2020.70.

9 J. Cook and D. Moshkovitz. Tighter MA/1 circuit lower bounds from verifier efficient PCPs for PSPACE. Technical Report TR22-014, ECCC, 2022.

10 I. Dinur, E. Fischer, G. Kindler, R. Raz, and S. Safra. PCP characterizations of NP: Toward a polynomially-small error-probability. *Computational Complexity*, 20(3):413–504, 2011.

11 I. Dinur and P. Harsha. Composition of low-error 2-query PCPs using decodable PCPs. In *Proc. 50th IEEE Symp. on Foundations of Computer Science*, pages 472–481, 2009.

12 I. Dinur, P. Harsha, and G. Kindler. Polynomially low error pcps with polyloglog n queries via modular composition. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proc. 47th ACM Symp. on Theory of Computing*, pages 267–276. ACM, 2015.

13 I. Dinur and D. Steurer. Analytical approach to parallel repetition. In *Proc. 46th ACM Symp. on Theory of Computing*, 2014.

14 U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996.

15 J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

16 A. Healy, S. P. Vadhan, and E. Viola. Using Nondeterminism to Amplify Hardness. *SIAM J. Comput.*, 35(4):903–931, 2006. doi:10.1137/S0097539705447281.

17 S. Hirahara. Non-Black-Box Worst-Case to Average-Case Reductions within NP. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 247–258, 2018. doi:10.1109/FOCS.2018.00032.

18 S. Hirahara. NP-Hardness of Learning Programs and Partial MCSP. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 968–979, 2022. doi:10.1109/FOCS54457.2022.00095.

19 S. Hirahara and O. Watanabe. Limits of Minimum Circuit Size Problem as Oracle. In *Proceedings of the Conference on Computational Complexity (CCC)*, pages 18:1–18:20, 2016. doi:10.4230/LIPIcs.CCC.2016.18.

20 J. M. Hitchcock and A. Pavan. On the NP-Completeness of the Minimum Circuit Size Problem. In *Proceedings of the Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 236–245, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.236.

21 J. Holmerin and S. Khot. A new PCP outer verifier with applications to homogeneous linear equations and max-bisection. In *Proc. 36th ACM Symp. on Theory of Computing*, pages 11–20, 2004.

22 R. Ilango. Constant Depth Formula and Partial Function Versions of MCSP are Hard. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 424–433, 2020. doi:10.1109/FOCS46700.2020.00047.

23 R. Impagliazzo and A. Wigderson. P = BPP if E Requires Exponential Circuits: Derandomizing the XOR Lemma. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, pages 220–229, 1997. doi:10.1145/258533.258590.

24 M. Ito, A. Saito, and T. Nishizeki. Multiple Assignment Scheme for Sharing Secret. *J. Cryptol.*, 6(1):15–20, 1993. doi:10.1007/BF02620229.

25 Z. Ji, A. Natarajan, T. Vidick, J. Wright, and H. Yuen. MIP* = RE. Submitted, 2020.

26 V. Kabanets and J. Cai. Circuit minimization problem. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 73–79, 2000. doi:10.1145/335305.335314.

27 J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proc. 32nd ACM Symp. on Theory of Computing*, pages 80–86, 2000.

28 S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-epsilon. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

29 S. Khot and N. K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into l_1. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 53–62, 2005.

30 J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 723–732, 1992.

31 A. R. Klivans and D. van Melkebeek. Graph Nonisomorphism Has Subexponential Size Proofs Unless the Polynomial-Time Hierarchy Collapses. *SIAM J. Comput.*, 31(5):1501–1526, 2002. doi:10.1137/S0097539700389652.

32 L. A. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

33 Y. Liu and R. Pass. On One-way Functions and Kolmogorov Complexity. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 1243–1254, 2020. doi:10.1109/FOCS46700.2020.00118.

34 D. Moshkovitz and R. Raz. Two query PCP with sub-constant error. *Journal of the ACM*, 57(5), 2010.

35 N. Nisan and A. Wigderson. Hardness vs Randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994. doi:10.1016/S0022-0000(05)80043-1.

36 I. C. Oliveira and R. Santhanam. Hardness Magnification for Natural Problems. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 65–76, 2018.

37 C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

38 O. Paradise. Smooth and strong pcps. *Comput. Complex.*, 30(1):1, 2021.

39 O. Reingold, S. P. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. *Annals of Mathematics*, 155(1):157–187, 2002.

40 H. Ren, R. Santhanam, and Z. Wang. On the Range Avoidance Problem for Circuits. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 640–650, 2022. doi:10.1109/FOCS54457.2022.00067.

41 M. Saks and R. Santhanam. Circuit Lower Bounds from NP-Hardness of MCSP Under Turing Reductions. In *Proceedings of the Computational Complexity Conference (CCC)*, pages 26:1–26:13, 2020. doi:10.4230/LIPIcs.CCC.2020.26.

42 R. Santhanam. Pseudorandomness and the Minimum Circuit Size Problem. In *Proceedings of the Innovations in Theoretical Computer Science Conference (ITCS)*, pages 68:1–68:26, 2020. doi:10.4230/LIPIcs.ITCS.2020.68.

43 R. Shaltiel and C. Umans. Pseudorandomness for Approximate Counting and Sampling. *Computational Complexity*, 15(4):298–341, 2006. doi:10.1007/s00037-007-0218-9.

44 L. Trevisan. Extractors and pseudorandom generators. *J. ACM*, 48(4):860–879, 2001. doi:10.1145/502090.502099.

45 D. Uhlig. On the synthesis of self-correcting schemes from functional elements with a small number of reliable elements. *Mathematical Notes of the Academy of Sciences of the USSR*, 15(6):558–562, 1974.

46 D. Uhlig. Networks Computing Boolean Functions for Multiple Input Values. In *Poceedings of the London Mathematical Society Symposium on Boolean Function Complexity*, pages 165–173, USA, 1992. Cambridge University Press.

47 S. P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.

48 I. Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987. URL: http://ls2-www.cs.uni-dortmund.de/monographs/bluebook/.

49 R. Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42:231–240, 2010.

Structural Parameterizations of b-Coloring

Lars Jaffke
University of Bergen, Norway

Paloma T. Lima
IT University of Copenhagen, Denmark

Roohani Sharma
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

The b-Coloring problem, which given a graph G and an integer k asks whether G has a proper k-coloring such that each color class has a vertex adjacent to all color classes except its own, is known to be FPT parameterized by the vertex cover number and XP and W[1]-hard parameterized by clique-width. Its complexity when parameterized by the treewidth of the input graph remained an open problem. We settle this question by showing that b-Coloring is XNLP-complete when parameterized by the pathwidth of the input graph. Besides determining the precise parameterized complexity of this problem, this implies that b-Coloring parameterized by pathwidth is W[t]-hard for all t, and resolves the parameterized complexity of b-Coloring parameterized by treewidth. We complement this result by showing that b-Coloring is FPT when parameterized by neighborhood diversity and by twin cover, two parameters that generalize vertex cover to more dense graphs, but are incomparable to pathwidth.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases b-coloring, structural parameterization, XNLP, pathwidth, neighborhood diversity, twin cover

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.40

Funding *Paloma T. Lima*: This work received funding from the Independent Research Fund Denmark grant agreement number 2098-00012B.

Acknowledgements We would like to thank an anonymous reviewer for carefully reading our paper and for many useful comments.

1 Introduction

A *b-coloring* of a graph G is a proper vertex-coloring such that each color class has a vertex, called *b-vertex*, that has a neighbor in each color class except its own. This problem originated in the study of the color-suppressing heuristic for the Graph Coloring problem: Start with any proper coloring of G, and keep on *suppressing* color classes as long as you can. Here, a color class C can be suppressed, if for each vertex with color C, there is a color $C' \neq C$ that does not yet appear in its neighborhood. This allows us to recolor all vertices in C and thereby lower the number of colors by one. The colorings which do not allow for further improvements are exactly the b-colorings, so the largest integer k such that a graph G admits a b-coloring with k colors determines the worst-case behaviour of this heuristic, when applied to G. This quantitity is referred to as the *b-chromatic number*. In this work, we study the following decision problem related to b-colorings. For more details on computational problems associated with b-colorings, we refer to [17, 18, 23].

b-Coloring

Input:	Graph G, integer k
Question:	Does G have a b-coloring with k colors?

This problem is known to be NP-complete, even when the number of colors is fixed [17]. The complexity of b-Coloring has been studied on several graph classes, see for instance [7, 8, 9, 10, 16, 17, 21, 24]. Recent work also considered first *structural parameterizations*: Jaffke, Lima, and Lokshtanov [18] showed that b-Coloring is FPT when parameterized by the vertex cover number of the input graph, and XP and W[1]-hard when parameterized by clique-width. Arguably the most prominent parameter (between the vertex cover number and the clique-width) is *treewidth*. Parameterized by treewidth plus the number of colors, b-Coloring is FPT [1, 18]. However, in the setting when the number of colors is part of the input, the parameterized complexity of b-Coloring by treewidth remained open. In fact, this problem has already been stated explicitly over a decade ago, see for instance [23]. Observe that even n-vertex forests can have b-colorings with $\sqrt{n}$ colors (think of a forest where each of the $\sqrt{n}$ components is a star with $\sqrt{n}-1$ leaves).

The first main result of this work is to resolve this open problem, showing hardness. We prove a stronger hardness result than W[1]-hardness by treewidth, namely XNLP-completeness by the more restrictive parameter pathwidth. The class XNLP has recently been coined by Bodlaender et al. [5], derived from earlier work of Elberfeld, Stockhusen, and Tantau [13], as a means of addressing the question of *completeness* of parameterized problems. XNLP is the class of parameterized problems that can be solved by a nondeterministic algorithm using $f(k) \cdot n^c$ time and $f(k) \cdot \log n$ space, where k is the parameter, n the input size, c a constant, and f a computable function. Several parameterized problems have been shown to be XNLP-complete [3, 4, 5], most relevant for our work problems parameterized by linear width measures [3, 4]. While XNLP-hardness reductions are often very similar to reductions proving W-hardness, they yield a much stronger result. As XNLP contains the entire W-hierarchy [5], XNLP-hardness implies W[t]-hardness for all $t \in \mathbb{N}$.

▶ **Theorem 1.** *b-Coloring parameterized by pathwidth is* XNLP*-complete, and therefore* W[t]*-hard for all $t \in \mathbb{N}$.*

Notice that the previous theorem implies that b-Coloring parameterized by treewidth or by clique-width is W[t]-hard for all $t \in \mathbb{N}$, therefore resolving the open question of the parameterized complexity of b-Coloring parameterized by treewidth [18, 23], and significantly strengthening the W[1]-hardness result by clique-width [18].

We complement Theorem 1 with two positive results about generalizations of the vertex cover number. The FPT-algorithm parameterized by vertex cover of [18] essentially follows from two observations. First, that the number of colors in any b-coloring of a graph with vertex cover number t is bounded by a function of t. Second, that the treewidth is always at most the vertex cover number. Therefore, the algorithm follows by an FPT-algorithm parameterized by treewidth plus number of colors. The generalizations we consider here, neighborhood diversity and twin-cover, both extend the vertex cover number to simply structured dense graphs; in particular, complete graphs have twin-cover number 0 and neighborhood diversity 1. This means that in both parameterizations, the number of colors in a b-coloring can be as high as $\Omega(n)$. Nevertheless, we obtain FPT-algorithms in both cases.

▶ **Theorem 2.** *b-Coloring parameterized by the twin-cover number or by the neighborhood diversity of a graph is fixed-parameter tractable.*

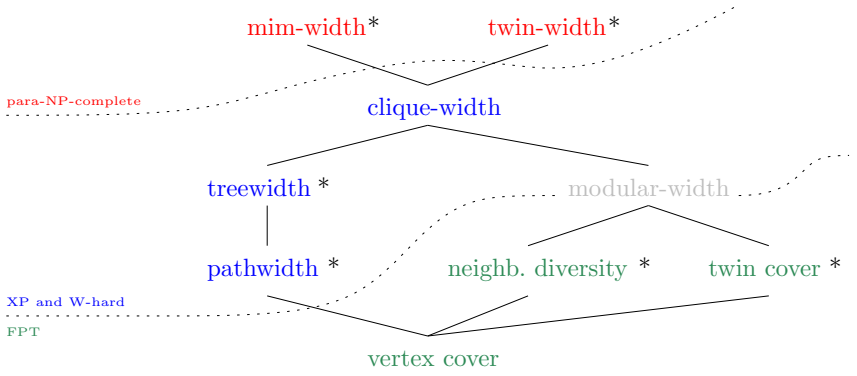


Figure 1 Known results about structural parameterizations of b-Coloring. Results marked with * can be found in this work. The complexity of b-Coloring parameterized by modular-width remains open.

Lastly, we observe by two trivial reductions that the XP-algorithms parameterized by clique-width cannot be extended to the more general width measures mim-width and twin-width. In both cases, this follows directly from known hardness results of Graph Coloring on certain graph classes. In the case of twin-width, this even holds when the number of colors is a fixed constant. This stronger hardness result does not hold for mim-width, as b-Coloring is expressible in DN logic by a sentence whose length depends only on the number of colors, and therefore XP parameterized by the mim-width of a given decomposition plus the number of colors [2].

▶ **Observation 3.** *b-Coloring is* NP*-complete on graphs of linear mim-width* 2*, and on graphs of twin-width at most* 8*. In the case of twin-width, para-*NP*-hardness even holds when the number of colors is any fixed constant $q \geq 3$.*

We summarize these results in Figure 1. Note that the parameterization modular-width, which is a common generalization of neighborhood diversity and twin-cover, remains open. The remainder of the paper is organized as follows. In Section 2, we give the necessary background and definitions, and justify Observation 3. In Section 3, we consider the parameterization by pathwidth and prove Theorem 1, and in Sections 4 and 5 we consider neighborhood diversity and twin-cover, respectively, to prove Theorem 2. We conclude in Section 6. Proofs of statements marked "★" can be found in the full version.

2 Preliminaries

Basic notations and definitions. For two integers $a \leq b$ we let $[a..b] = \{a, a+1, \ldots, b\}$, and for a positive integer a, we let $[a] = [1..a]$. All graphs considered here are finite and simple. For an (undirected or directed) graph G, we denote its vertex set by $V(G)$ and its edge set by $E(G)$. For an edge $\{u, v\} \in E(G)$, we use the shorthand "uv". If G is a directed graph, then denoting the edge $e = (u, v) \in E(G)$ by uv also points to e being directed from u to v. Given an undirected graph G, an *orientation* of G, denoted by $\overrightarrow{G}$, is a directed graph obtained from G by replacing each edge $\{u, v\} \in E(G)$ by either (u, v) or (v, u). The *neighborhood of a vertex* v is defined as $N(v) = \{u \in V(G) \mid uv \in E(G)\}$. The *closed neighborhood of* v is defined as $N[v] = N(v) \cup \{v\}$. The *neighborhood of a set* $S \subseteq V(G)$ is defined similarly, that is, $N(S) = \{u \in V(G) \setminus S \mid us \in E(G) \text{ for some } s \in S\}$. The *closed neighborhood of* S is $N[S] = N(S) \cup S$. A set $S \subseteq V(G)$ is *independent* if for all pairs of distinct $u, v \in S$,

$uv \notin E(G)$. A set $C \subseteq V(G)$ is a *clique* if for all pairs of distinct $u, v \in C$, $uv \in E(G)$. A *star* is an undirected graph with one special vertex called the *center* that is adjacent to all of the remaining vertices, called *leaves*, which form an independent set.

Colorings. A *proper coloring with k colors* of a graph G is a partition of $V(G)$ into k independent sets, called *color classes*. For a graph G and a proper coloring of G, a vertex $v \in V(G)$ is a *b-vertex* if it has a neighbor in all color classes except its own. A *b-coloring with k colors* of a graph G is a proper coloring of G such that each color class contains a b-vertex. We call such a coloring a *k-b-coloring*. In this work we consider the following problem.

b-Coloring

Input:	Graph G, integer k
Question:	Does G have a b-coloring with k colors?

2.1 Width measures

We now define the width measures relevant for this work and state some known facts about them for completeness.

▶ **Definition 4** (Module, Modular Partition, Quotient Graph). *A* module *of a graph G is a set of vertices $M \subseteq V(G)$ such that for all $v \in V(G) \setminus M$, either $M \subseteq N_G(v)$ or $M \cap N_G(v) = \emptyset$. A partition $\mathcal{P}$ of $V(G)$ is a* modular partition *if all parts of $\mathcal{P}$ are modules in G. The* quotient graph *of $\mathcal{P}$, denoted by $G/\mathcal{P}$ is the graph whose vertex set is $\mathcal{P}$ such that for all $P, Q \in \mathcal{P}$, $PQ \in E(G/\mathcal{P})$ if P is complete to Q in G and $PQ \notin E(G/\mathcal{P})$ if P is anti-complete to Q in G.*

▶ **Definition 5** (Neighborhood Diversity [22]). *An* ND-partition *of a graph G is a modular partition $\mathcal{P}$ of $V(G)$ such that each part of $\mathcal{P}$ is either a clique (called a* clique part*) or an independent set (called* independent part*) in G. The* neighborhood diversity *of a graph G is the minimum number of parts in any ND-partition of G.*

▶ Remark 6. Note that the neighborhood diversity can also be defined as follows: for a graph G, say that two vertices u, v are equivalent if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. Each equivalence class consists of a complete part and an independent part which gives the optimal ND-partition in polynomial time.

▶ **Definition 7** (Twin Cover [15]). *A set $S \subseteq V(G)$ of a graph G is a* twin cover*, if for each edge $uv \in V(G)$, either (i) $\{u, v\} \cap S \neq \emptyset$, or (ii) or u and v are twins in G. The* twin cover number *of G is the smallest size of any twin cover of G.*

▶ **Observation 8.** *If S is a twin-cover of a graph G then each connected component of $G - S$ is a clique consisting of twins.*

Let G be a graph and S a vertex cover of size k. It is clear that S is also a twin cover. Moreover, for each $A \subseteq S$, let $P_A \subseteq V(G) \setminus S$ be the set of all vertices v with $N(v) = A$. Then, the partition of $V(G)$ consisting of all singletons of S plus the sets P_A for all $A \subseteq S$ is an ND-partition of G, so G has neighborhood diversity at most $k + 2^k$ [22].

Neighborhood diversity and twin cover number are incomparable: consider for instance $K_{n,n}$, a complete bipartite graph with n vertices on each side. The neighborhood diversity of $K_{n,n}$ is two, as the natural bipartition of its vertices is an ND-partition. On the other hand, each twin cover of $K_{n,n}$ has size at least n (it has to fully contain one of the sides).

Conversely, consider the windmill graph W_n with n petals, that is, a collection of n triangles where each triangle has one special vertex that is identified with all other special vertices. The twin cover number of W_n is one (just take the vertex resulting from identifying all the special vertices), while the neighborhood diversity of W_n is $n+1$. For the lower bound, observe that no two non-special vertices from distinct triangles can be in the same part of an ND-partition.

▶ **Definition 9.** *Let G be a graph. A* path decomposition *of G is a sequence $\mathcal{B} = B_1, \ldots, B_d$ of subsets of $V(G)$ called* bags *covering $V(G)$ such that:*
1. *For each edge $e \in E(G)$, there is some $i \in [d]$ such that $e \subseteq B_i$.*
2. *For each $h, i, j \in [d]$ with $h < i < j$, $B_h \cap B_j \subseteq B_i$.*

The width *of $\mathcal{B}$ is $\max_{i\in[d]}|B_i| - 1$, and the* pathwidth *of G is the smallest width of all its path decompositions.*

Membership in XNLP of b-Coloring parameterized by pathwidth will follow from the membership of b-Coloring parameterized by a linear width measure with more expressive power than pathwidth, namely one that is equivalent to linear clique-width. We define it next and show its relation to pathwidth.

▶ **Definition 10.** *Let G be a graph and $S \subseteq V(G)$. The* module number *of S is the number of equivalence classes of the equivalence relation $\sim_S$ defined as: $u \sim_S v \Leftrightarrow N(u) \cap (V(G) \setminus S) = N(v) \cap (V(G) \setminus S)$. Let $\pi = v_1, \ldots, v_n$ be a linear order of $V(G)$. The* module-width *of π is the maximum, over all i, of the module number of $\{v_1, \ldots, v_i\}$. The* linear module-width *of G is the minimum module-width over all its linear orders.*

▶ **Lemma 11 (★).** *Let G be a graph and $\mathcal{B}$ be a path decomposition of G of width w. Then one can construct in polynomial time and logarithmic space a linear order of module-width at most $w + 2$.*

2.1.1 Linear mim-width

For a graph G and a linear order $\lambda = v_1, \ldots, v_n$ of $V(G)$, the *mim-width of λ* is the maximum, over all $i \in [n-1]$, of the size of an induced matching in the bipartite subgraph of G consisting of all edges that have one endpoint in $\{v_1, \ldots, v_i\}$ and the other in $\{v_{i+1}, \ldots, v_n\}$. The *linear mim-width* of a graph G is the minimum mim-width over all its linear orders. We observe that b-Coloring parameterized by linear mim-width is para-NP-complete.

▶ **Observation 12 (★).** *b-Coloring is NP-complete on graphs of linear mim-width 2.*

2.1.2 Twin-width

We skip the definition of twin-width here, and refer the reader to [6].

▶ **Observation 13 (★).** *For any $k \geq 3$, the problem of determining if a graph G has a b-coloring with k colors is NP-complete on graphs of twin-width at most 8.*

2.2 The class XNLP

We assume familiarity with the basic technical notions of parameterized complexity and refer to [12] for an overview. The class XNLP, introduced as $N[f\ \text{poly}, f\ \log]$ by Elberfeld et al. [13], consists of the parameterized decision problems that given an n-bit input with

parameter k can be solved by a non-deterministic algorithm that simultaneously uses at most $f(k)n^c$ time and at most $f(k)\log n$ space, where f is a computable function and c is a constant. We refer to [5, 13] for more details on this complexity class. Hardness in XNLP can be transferred via *parameterized logspace reductions* [13] which are parameterized reductions in the traditional sense [12] with the additional constraint of using only $f(k) + \mathcal{O}(\log n)$ space, where once again k is the parameter of the problem and n is the input size.

3 Pathwidth

We show b-Coloring is XNLP-complete via a reduction from the following problem which is known to be XNLP-complete when parameterized by the width of a given path decomposition of the input graph [3].

Circulating Orientation

Input:	Undirected graph G with edge weights $\mathbf{w}\colon E(G) \to \mathbb{N}$ given in unary.
Question:	Is there an orientation $\overrightarrow{G}$ of G such that for every vertex $v \in V(G)$: $\sum_{vx \in E(\overrightarrow{G})} \mathbf{w}(vx) = \sum_{xv \in E(\overrightarrow{G})} \mathbf{w}(xv)$?

▶ **Theorem 14.** *b-Coloring parameterized by the width of a given path decomposition of the input graph is* XNLP*-complete.*

Proof. To show XNLP-hardness, we give a parameterized logspace-reduction from the Circulating Orientation problem parameterized by the width of a given path decomposition of the input graph, which was shown to be XNLP-complete in [3]. Let $(G, \mathbf{w})$ be an instance of Circulating Orientation, given with a path decomposition $\mathcal{B}$ of G. We let $n = |V(G)|$, $m = |E(G)|$, and $\mathbf{W} = \sum_{e \in E(G)} \mathbf{w}(e)$. For each vertex $v \in V(G)$, we let $W_v = \sum_{uv \in E(G)} \mathbf{w}(uv)$. We may assume that G is connected and that for all $e \in E(G)$, $\mathbf{w}(e) \geq 1$; therefore $\mathbf{W} \geq m \geq n - 1$.

We construct an equivalent instance (H, k) of b-Coloring. We let

$$k = 2\mathbf{W} + 3m + n + 2. \tag{1}$$

We begin the construction of H which is illustrated in Figure 2 by adding $2\mathbf{W} + 2$ disjoint copies of a star with $k - 1$ leaves. Let $S^\star$ be one of these stars. We denote its center by $s^\star$ and refer to it throughout the proof as the *superstar*. The remaining ones are referred to as *anonymous*. We partition a subset of the leaves of $S^\star$ into $\mathcal{L} = \{L_{e,v} \mid e \in E(G), v \in e\}$ where for all $e \in E(G)$ and $v \in e$, $|L_{e,v}| = \mathbf{w}(e)$. Note that this is possible since $k - 1 \geq 2\mathbf{W}$.

Vertex gadget. For each $v \in V(G)$, we add v, as well as a set P_v of $k - \frac{3}{2}W_v - 1$ independent vertices to H. We add all edges between v and P_v. Furthermore, for each edge $e \in E(G)$ such that $v \in e$, we connect v and the vertices in $L_{e,v}$ in H.

Edge gadget. For each $e = uv \in E(G)$, we add the following gadget to H. First, it has two vertices $x_{e,u}$ and $x_{e,v}$, a set Y_e of $\mathbf{w}(e)$ vertices, and a set Z_e of $k - 2\mathbf{w}(e) - 3$ vertices. The vertex $x_{e,u}$ is adjacent to $Y_e \cup Z_e \cup L_{e,u}$, and $x_{e,v}$ is adjacent to $Y_e \cup Z_e \cup L_{e,v}$. We make u and v adjacent to Y_e. We furthermore add two new vertices $q_{e,1}$ and $q_{e,2}$ to H that are connected by an edge, as well as all edges between $q_{e,h}$ and $Z_e \cup L_{e,u} \cup L_{e,v} \cup \{x_{e,u}, x_{e,v}\}$ for all $h \in [2]$. We let $X = \{x_{e,u}, x_{e,v} \mid e = uv \in E(G)\}$, and $\mathcal{Q} = \{q_{e,1}, q_{e,2} \mid e \in E(G)\}$.

Adding all vertex and edge gadgets finishes the construction of H, which can be performed using only logarithmic space.

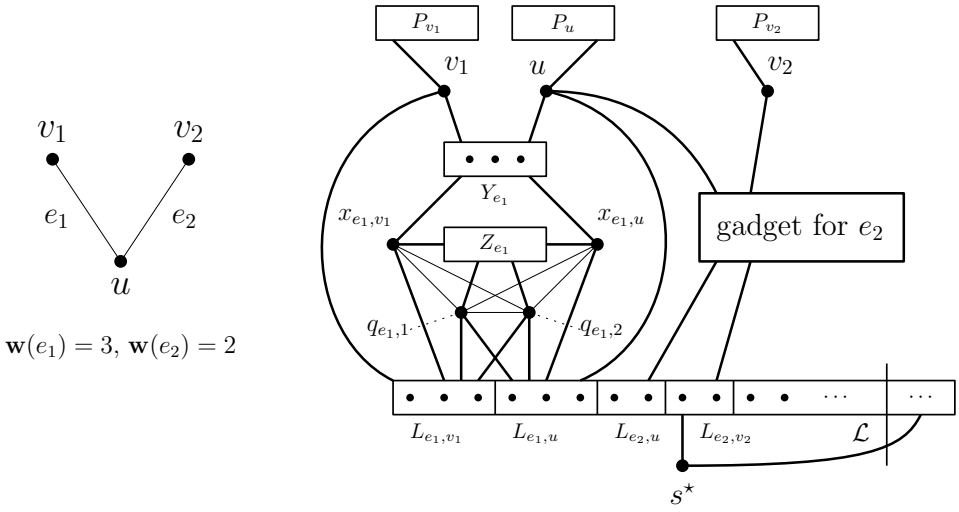


Figure 2 Sketch of the main part of the reduction. Bold edges mean that all edges between the corresponding sets are present. All vertex sets represented by single boxes are independent. Note that $|L_{e,v_1}| = |L_{e,u}| = |Y_{e_1}| = \mathbf{w}(e_1) = 3$ and recall that $|Z_{e_1}| = k - 2\mathbf{w}(e_1) - 3$.

▷ Claim 15. If $(G, \mathbf{w})$ has a circulating orientation, then H has a b-coloring with k colors.

Proof. Let $\overrightarrow{G}$ be the circulating orientation of $(G, \mathbf{w})$. We give a coloring of the vertices of H with colors $[0..(k-1)]$. To do so, we identify some important subsets of $[0..(k-1)]$ whose b-vertices will appear in targeted regions of H. First, we let $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$. We construct a proper coloring of H such that once the coloring is completed, the following hold.

1. The vertex $s^\star$ (the center of the superstar) is a b-vertex of color 0.
2. For each $i \in [n]$, v_i is a b-vertex of color i.
3. For each $i \in [m]$, $q_{e_i,1}$ is a b-vertex of color $n+i$, and $q_{e_i,2}$ is a b-vertex of color $m+n+i$.
4. For each $i \in [m]$, either $x_{e_i,u}$ or $x_{e_i,v}$, where $e_i = uv$, is a b-vertex of color $2m+n+i$.
5. Each of the remaining $k - (3m+n+1) = 2\mathbf{W}+1$ colors has a b-vertex that is a center of an anonymous star.

Let $S_1, \ldots, S_{2\mathbf{W}+1}$ be the anonymous stars with centers $s_1, \ldots, s_{2\mathbf{W}+1}$, respectively. For each $i \in [2\mathbf{W}+1]$, we assign s_i the color $3m+n+i$, and the leaves of S_i the colors $[0..(k-1)] \setminus \{3m+n+i\}$. This satisfies Item 5. We assign $s^\star$ the color 0 and its leaves the colors $[k-1]$ in such a way that colors $[(3m+n+1)..(3m+n+2\mathbf{W})]$ appear on the vertices in $\mathcal{L}$. This satisfies Item 1. For each $i \in [n]$, we assign v_i color i. For each $i \in [m]$ and each $v \in e_i$, we let $C_{e_i,v}$ be the colors appearing on $L_{e_i,v}$, and we assign $x_{e_i,v}$ the color $2m+n+i$.

We now color the edge gadgets. Let $i \in [m]$ and $e_i = uv$. We give $q_{e_i,1}$ color $n+i$ and $q_{e_i,2}$ color $m+n+i$. We assign the vertices in Z_{e_i} the colors

$$[0..(k-1)] \setminus (C_{e_i,u} \cup C_{e_i,v} \cup \{n+i, m+n+i, 2m+n+i\}).$$

If e_i is directed from u to v in $\overrightarrow{G}$, then we repeat colors $C_{e_i,u}$ on Y_{e_i}. Observe that this makes $x_{e_i,v}$ a b-vertex for color $2m+n+i$: it sees colors $C_{e_i,v}$ on $L_{e_i,v}$, colors $C_{e_i,u}$ on Y_{e_i}, and the remaining colors other than its own on $Z_{e_i} \cup \{q_{e_i,1}, q_{e_i,2}\}$. Moreover, $q_{e_i,1}$ is a b-vertex for color $n+i$, since it sees color $m+n+i$ on $q_{e_i,2}$, color $2m+n+i$ on $x_{e_i,v}$, and the remaining colors on $L_{e_i,u} \cup L_{e_i,v} \cup Z_{e_i}$. Similarly, $q_{e_i,2}$ is a b-vertex for color $m+n+i$. Once this is done for all i, Items 3 and 4 are satisfied.

We now color the vertex gadgets. We first argue that each $v \in V(G)$ already sees precisely $\frac{3}{2}W_v$ colors in its neighborhood. This is because v sees W_v colors on $\bigcup_{e \in E(G), v \in e} L_{e,v}$, and for each edge e that is directed towards v, there are $\mathbf{w}(e)$ additional colors appearing in the neighborhood of v; concretely, on the set Y_e of the corresponding edge gadget. Since $\overrightarrow{G}$ is circulating, the latter contribute with an additional $\frac{1}{2}W_v$ colors in total. Therefore, we can distribute the remaining $k - \frac{3}{2}W_v - 1$ colors on the set P_v, which makes v a b-vertex. This satisfies Item 2, and we have arrived at a b-coloring of H with k colors. ◁

We now work towards the reverse implication of the correctness proof. We start with a claim regarding the location of the b-vertices in any b-coloring of H with k colors. Throughout the following, we denote by A the set of centers of the anonymous stars.

▷ Claim 16. Each b-coloring of H with k colors has precisely one b-vertex per color. Moreover, the b-vertices are $\{s^\star\} \cup V(G) \cup \mathcal{Q} \cup A$, and for each $e = uv \in E(G)$, precisely one of $x_{e,u}$ and $x_{e,v}$.

Proof. The only vertices with high enough degree (at least $k-1$) to become b-vertices in such a coloring of H are in $\{s^\star\} \cup V(G) \cup \mathcal{Q} \cup A \cup X$. Note that this set has size $2\mathbf{W} + 4m + n + 2 = k + m$.

We argue that the gadget of each edge $e = uv$ can contain at most three b-vertices. Note that only four of its vertices, $x_{e,u}$, $x_{e,v}$, $q_{e,1}$, and $q_{e,2}$ have high enough degree to be b-vertices. Suppose for a contradiction that $x_{e,u}$ and $x_{e,v}$ are b-vertices for colors c_u and c_v, respectively, where $c_u \neq c_v$. For $x_{e,u}$ to be a b-vertex of color c_u, it needs to have a neighbor colored c_v. By the structure of H, this vertex has to be contained in $L_{e,u}$. Similarly, we can conclude that $L_{e,v}$ contains a vertex colored c_u. But this means that both $q_{e,1}$ and $q_{e,2}$ have two neighbors colored c_u and two neighbors colored c_v. Since $\deg_H(q_{e,h}) = 2\mathbf{w}(e) + |Z_e| + 3 = k$ for all $h \in [2]$, this means that each of these vertices sees at most $k-2$ colors in its neighborhood, so neither of them is a b-vertex. Therefore we can assume from now on that $x_{e,u}$ and $x_{e,v}$ receive the same color.

Since we only have $k + m$ vertices of high enough degree to be b-vertices, we can only have enough b-vertices if each edge gadget has exactly three b-vertices, and if all vertices in $\{s^\star\} \cup V(G) \cup A$ are b-vertices. Now suppose that for some edge $e = uv \in E(G)$, both $x_{e,u}$ and $x_{e,v}$ are b-vertices for their color. By the structure of H, this implies that the same colors have to appear on $L_{e,u}$ and $L_{e,v}$. But $s^\star$ needs to be a b-vertex, now there are $\mathbf{w}(e) \geq 1$ colors in its neighborhood that repeat. Since $\deg_H(s^\star) = k - 1$, this is not possible. This yields the claim. ◁

Throughout the following, we assume that we have a b-coloring of H with k colors. Again, for each $e \in E(G)$ and $v \in e$, we denote by $C_{e,v}$ the set of colors appearing on the vertices $L_{e,v}$. We prove another auxiliary claim.

▷ Claim 17.

1. For each $e, e' \in E(G)$ and $v \in e$, $v' \in e'$, if $(e, v) \neq (e', v')$, then $C_{e,v} \cap C_{e',v'} = \emptyset$.
2. For each $e = uv \in E(G)$, either colors $C_{e,u}$ or colors $C_{e,v}$ appear on Y_e; the former if $x_{e,v}$ is a b-vertex and the latter if $x_{e,u}$ is a b-vertex.

Proof. Item 1. By Claim 16, we know $s^\star$ is a b-vertex. Since its degree is $k-1$, all its neighbors must receive distinct colors. Hence Item 1 follows.

Item 2. By Claim 16, either $x_{e,v}$ or $x_{e,u}$ is a b-vertex for its color. Suppose that $x_{e,v}$ is a b-vertex for color i (the other case is analogous). For $x_{e,v}$ to be a b-vertex, the colors $C_{e,u}$ have to appear in its neighborhood. We show that the colors $C_{e,u}$ have to appear on Y_e,

which yields the claim. By Claim 17 Item 1, we have that $C_{e,u} \cap C_{e,v} = \emptyset$, so the colors $C_{e,u}$ have to appear on $Z_e \cup Y_e$. We rule out that they appear on Z_e. For the following argument, recall that by Claim 16, $q_{e,1}$ is a b-vertex for its color; moreover, its degree is k, so it sees exactly one color twice in its neighborhood.

We distinguish two cases based on the color of the vertex $x_{e,u}$. Suppose $x_{e,u}$ received color i as well. Then, $q_{e,1}$ sees color i twice, meaning that all remaining colors appear exactly once on its neighborhood. Since $Z_e \cup L_{e,u} \subset N(q_{e,1}) \setminus \{x_{e,u}, x_{e,v}\}$, no color from $C_{e,u}$ appears on Z_e. Now suppose that $x_{e,u}$ received a color $j \neq i$. Since $x_{e,v}$ is a b-vertex for color i, and since $\deg(x_{e,v}) = k - 1$, there is precisely one vertex with color j in $N(x_{e,v})$. Since the given coloring of H is proper, this vertex cannot be in $Y_e \cup Z_e \cup \{q_{e,1}, q_{e,2}\} \subseteq N(x_{e,u})$. Therefore, there is a vertex of color j in $L_{e,v}$. This means that $q_{e,1}$ sees color j twice, once on $x_{e,u}$ and once on a vertex in $L_{e,v}$. Subsequently, the vertices in $L_{e,u} \cup Z_e$ all receive unique colors, implying once again that no color from $C_{e,u}$ appears on Z_e. In either case, the only way that $x_{e,v}$ sees colors $C_{e,u}$ is if they appear on Y_e, which proves the claim. ◁

We now construct an orientation $\overrightarrow{G}$ of G. For each edge $e = uv \in E(G)$, if $x_{e,u}$ is a b-vertex, then we orient e towards u, and if $x_{e,v}$ is a b-vertex, we orient e towards v. Note that by Claim 16, this is well-defined. Throughout the following whenever we write "uv" for an edge in $\overrightarrow{G}$, we mean that the edge uv is directed from u to v in $\overrightarrow{G}$. The next claim completes the correctness proof of the reduction.

▷ Claim 18. For each $v \in V(G)$, $\sum_{uv \in E(\overrightarrow{G})} \mathbf{w}(uv) = \frac{1}{2} W_v$.

Proof. We first show that $\sum_{uv \in E(\overrightarrow{G})} \mathbf{w}(uv) \geq \frac{1}{2} W_v$. By Claim 16, v is a b-vertex. Moreover, $\deg_H(v) = k + \frac{1}{2} W_v - 1$ since v has $k - \frac{3}{2} W_v - 1$ neighbors in P_v, W_v additional neighbors in the edge gadgets, W_v additional neighbours in $\mathcal{L}$, and no other neighbors. This means that for v to be a b-vertex, v needs to see at least $\frac{1}{2} W_v$ colors in $\bigcup_{e \in E(G), v \in e} Y_e$. Claim 17 then implies that there is a set of edges $\{e_1, \ldots, e_d\}$ incident with v and with $\sum_{i \in [d]} \mathbf{w}(e_i) \geq \frac{1}{2} W_v$ such that for all $i \in [d]$, $x_{e_i,v}$ is a b-vertex. This implies the inequality by our construction of $\overrightarrow{G}$.

Now we show that $\sum_{uv \in E(\overrightarrow{G})} \mathbf{w}(uv) \leq \frac{1}{2} W_v$. Let $\mathcal{Y} = \bigcup_{e \in E(G)} Y_e$, note that $|\mathcal{Y}| = \mathbf{W}$, and that to make each $v \in V(G)$ a b-vertex, $\frac{1}{2} W_v$ colors must appear in $N_H(v) \cap \mathcal{Y}$ that are not in $N_H(v) \setminus \mathcal{Y}$. Moreover, for each $e = uv \in E(G)$, Y_e has colors that appear in $N_H(u) \setminus \mathcal{Y}$ but not in $N_H(v) \setminus \mathcal{Y}$ or vice versa by Claim 17. Since $\mathbf{W} = \sum_{e \in E(G)} \mathbf{w}(e) = \sum_{v \in V(G)} \frac{1}{2} W_v$, we can conclude that if for some $v \in V(G)$, $\sum_{uv \in E(\overrightarrow{G})} \mathbf{w}(uv) > \frac{1}{2} W_v$, then there is another $v' \in V(G) \setminus \{v\}$ with $\sum_{uv' \in E(\overrightarrow{G})} \mathbf{w}(uv') < \frac{1}{2} W_{v'}$, contradicting the previous paragraph. ◁

▷ Claim 19. Given a path decomposition of G of width w, one can construct a path decomposition of H of width at most $w + 6$ in polynomial time and logarithmic space.

Proof. Let $\mathcal{B}$ be a path decomposition of G of width w. We add $s^\star$ to all bags of $\mathcal{B}$. For each vertex $v \in V(G)$, let $B_v \in \mathcal{B}$ be a bag containing v. We insert a sequence of $|P_v|$ bags after B_v containing B_v, and a unique vertex of P_v. For each edge $e = uv \in E(G)$, let B_e be a bag in $\mathcal{B}$ containing u and v. We insert a sequence of $|Y_e \cup Z_e \cup L_{e,u} \cup L_{e,v}|$ bags after B_e containing B_e, $x_{e,u}$, $x_{e,v}$, $q_{e,1}$, $q_{e,2}$, and a unique vertex of $L_{e,u} \cup L_{e,v} \cup Y_e \cup Z_e$. Finally, we append a sequence of bags forming a width-1 path decomposition of the anonymous stars. Note that this gives a path decomposition of H and there is no bag to which we added more than six vertices. It is easy to see that these operations can be performed within the claimed time and space requirements. ◁

Adapting the XP-algorithm for b-Coloring parameterized by module-width w [18] to a nondeterministic FPT-time and $f(w)\log n$ space algorithm, we can show that b-Coloring parameterized by linear module-width, and therefore by pathwidth, belongs to XNLP. This can be done similarly as in the case of Graph Coloring parameterized by linear clique-width as shown in [4]. This is summarized in the statement below.

▷ Claim 20. b-Coloring parameterized by the module-width of a given linear order of the vertices of the input graph is in XNLP.

Membership then follows from Lemma 11 and Claim 20. This concludes the proof of the theorem. ◀

4 Neighborhood Diversity

In this section, we consider the parameterization by neighborhood diversity. We follow the same strategy as the one that Koutecký [20] applied for the Graph Coloring problem, that is, we give an ILP-formulation that can be solved efficiently by a parameterized ILP-algorithm due to Jansen and Rohwedder [19].[1]

▶ **Theorem 21** (Jansen and Rohwedder [19]). *For $A \in \mathbb{Z}^{r\times n}$, $b \in \mathbb{Z}^r$, $c \in \mathbb{Z}^n$, the ILP*

$$\min\{c^T x \colon Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$$

can be solved in time $\mathcal{O}((\sqrt{r}\Delta)^{2r}) \cdot \log \|b\|_\infty + \mathcal{O}(rn)$, where $\Delta = \|A\|_\infty = \max_{i,j} A(i,j)$ and $\|b\|_\infty = \max_i b(i)$.

Note that in the previous theorem, the number of rows r in the ILP is equal to the number of constraints. Recall that an optimal ND-partition can be computed in polynomial time (Remark 6).

▶ **Theorem 22.** *b-Coloring parameterized by the neighborhood diversity d of the input n-vertex graph is fixed-parameter tractable. Given an ND-partition of the input graph, the algorithm runs in time $2^{\mathcal{O}(d\log d)}\log n + \mathcal{O}(n)$.*

Proof. Suppose we want to find a b-coloring with k colors. Let G be a graph of neighborhood diversity at most d with ND-partition $\mathcal{P} = (P_1, \ldots, P_d)$. We create another partition $\mathcal{P}'$ of $V(G)$ as follows. For each P_i that is an independent set of size at least two, we pick one vertex v_i, let $P_i' = P_i \setminus \{v_i\}$, remove P_i from $\mathcal{P}$ and add P_i' and $\{v_i\}$ to $\mathcal{P}'$. All other parts of $\mathcal{P}$ are added to $\mathcal{P}'$ as they are. As a convention, we consider each $\{v_i\}$ a clique of size 1, and each such part in $\mathcal{P}'$ a clique part. Note that $d' = |\mathcal{P}'| \leq 2d$.

We start with a few observations.

1. If $u, v \in V(G)$ are false twins, then in each proper coloring of G, either both u and v are b-vertices for the same color, or neither of them is a b-vertex.
2. For each $P \in \mathcal{P}'$ that is a clique, either all vertices in P are b-vertices for their color, or none of them are.
3. In each b-coloring of G, each color class has a b-vertex contained in a clique part of $\mathcal{P}'$.

[1] Note that the arXiv-version contains an improved running time over the version published in the ITCS 2019 proceedings.

Item 1 is immediate, Item 2 follows from the fact that all vertices that are in the same part are twins, and Item 3 follows from Item 1 and our construction: if there was an independent part with more than one vertex in $\mathcal{P}$, we split off a single vertex into a new part, which is now considered a clique part. If in a b-coloring, some independent part (of the original ND-partition $\mathcal{P}$) had a b-vertex, then the split off vertex is a b-vertex for the same color by Item 1, considered a clique part in $\mathcal{P}'$.

Next, we guess which clique parts of $\mathcal{P}' = (P_1, \dots P'_{d'})$ contain b-vertices in the solution we are looking for. From now on, fix one such choice $B \subseteq [d']$.

We construct an ILP as follows. Let $H = G/\mathcal{P}'$. Each color class is described by its *type*, that is, the parts of $\mathcal{P}'$ it intersects. Note that each type is an independent set in H. Therefore, for each independent set I in H, we add a variable x_I, which counts how many color classes of that type there are. From now on, we denote by $\mathcal{I}(H)$ the independent sets of H. Now, the sum, over all independent sets I of H of the x_I will correspond to the total number of colors used. We add a constraint that ensures that this number is k. Moreover, for each clique part P'_i, we have to make sure that exactly $|P'_i|$ colors appear on that part, and in each independent part $P'_{i'}$, at least one color must appear. Finally, we have to ensure that there are k b-vertices. Note that, since all b-vertices are clique parts, by Observation 2, each vertex in each part P'_i for $i \in B$, has to be a b-vertex. Therefore, for each $i \in B$, we ensure that the number of colors intersecting the closed neighborhood of vertex i in H is equal to k. To ensure that each color class has a b-vertex, we use the objective function to minimize the number of color classes that do not intersect B. If this value is 0, then we have a b-coloring, otherwise not. The ILP is:

$$
\begin{aligned}
\min \quad & \sum_{I \in \mathcal{I}(H), I \cap B = \emptyset} x_I \\
\text{s.t.} \quad & \sum_{I \in \mathcal{I}(H)} x_I = k \\
& \sum_{I \in \mathcal{I}(H), i \in I} x_I = |P'_i|, && \text{if } P'_i \text{ is a clique} \\
& \sum_{I \in \mathcal{I}(H), i \in I} x_I \geq 1, && \text{if } P'_i \text{ is an independent set} \\
& \sum_{I \in \mathcal{I}(H), I \cap N_H[i] \neq \emptyset} x_I = k, && \text{if } i \in B
\end{aligned}
\tag{2}
$$

The correctness of this formulation follows fairly straightforwardly from the discussion above.

▶ **Observation 23.** *The previous ILP has a solution with value* 0 *if and only if* G *has a* b*-coloring with* k *colors whose* b*-vertices intersect precisely the parts* $\{P'_i \mid i \in B\}$.

For each guess of B, we construct an ILP as above. If there is one guess for which we have a solution with value 0, we report that G has a b-coloring with k colors, and say No otherwise. Correctness directly follows from Observation 23.

Let us analyze the run time. We can obtain the ND-partition $\mathcal{P}'$ from $\mathcal{P}$ in time $\mathcal{O}(n)$. We can then compress $\mathcal{P}'$ to remember only $|P'_i|$ for each $i \in [d']$ and one representative vertex per P'_i, also in time $\mathcal{O}(n)$. We solve $2^{\mathcal{O}(d)}$ many ILPs using Theorem 21 with $\mathcal{O}(d)$ rows and $2^{\mathcal{O}(d)}$ variables. From the compressed representation, each such ILP can be constructed in $2^{\mathcal{O}(d)} \log n$ time. Note that the inequalities (2) can be turned into equalities by adding at most $2^{\mathcal{O}(d)}$ slack variables. In the resulting ILP, the largest coefficient of any variable is 1, and the largest value on the right-hand side is at most n, since we may assume that $k \leq n$, and clearly, for each $i \in [d']$, $|P'_i| \leq n$. Therefore, we have $\Delta = 1$ and $\|b\|_\infty \leq n$, and each of the ILPs can be solved in time $2^{\mathcal{O}(d \log d)} \log n + d \cdot 2^{\mathcal{O}(d)} = 2^{\mathcal{O}(d \log d)} \log n$, yielding the claimed run time bound. ◀

5 Twin Cover

In this section, we prove the following theorem. This will be done by reducing the input graph of bounded twin cover number to a graph of bounded neighborhood diversity and then applying the algorithm from Theorem 22.

▶ **Theorem 24.** *b-Coloring parameterized by the twin cover number of the input graph is fixed-parameter tractable. Given a graph with n vertices, m edges, and twin cover number t, it is solvable in $2^{2^{\mathcal{O}(t)}}n + \mathcal{O}(m)$ time.*

Let (G, k) be an instance of b-Coloring. If the size of a minimum twin-cover of G is at most t, then one can compute a twin-cover S of G of size at most t in $\mathcal{O}(1.2378^t + tn + m)$ time [15]. Recall that each connected component of $G - S$ consists of a clique C consisting of twins (Observation 8). Since C is a clique, we may assume $|C| \leq k$.

Without loss of generality, let the color set be $\{1, \ldots, k\}$ and assume the vertices of S get colors from the set $\{1, \ldots, \min\{k, t\}\}$. For each $\texttt{col} : S \to \{1, \ldots, \min\{k, t\}\}$ such that $\texttt{col}$ is a proper coloring of $G[S]$, perform the following steps. Note, in the following, $\texttt{col}$ is fixed.

For each $A \subset S$, let $\mathcal{C}_A = \{C : C \text{ is a maximal clique of } G - S \text{ and } N(C) = A\}$. Throughout, we use the shorthand $\bigcup \mathcal{C}_A$ for $\bigcup_{C \in \mathcal{C}_A} C$. Let $c_A^{\texttt{col}}$ denote the number of distinct colors used by the vertices of A in $\texttt{col}$, that is $c_A^{\texttt{col}} = |\cup_{a \in A} \texttt{col}(a)|$. Note that if a b-coloring with k colors of G coincides with $\texttt{col}$ on the vertex set S, then for any $C \in \mathcal{C}_A$, $k \geq |C| + c_A^{\texttt{col}}$. Thus, if this inequality does not hold, then $\texttt{col}$ cannot be extended into a k-b-coloring of G. In this case, discard $\texttt{col}$ and consider the next available (non-discarded) proper coloring function on S. Henceforth, assume that for each $C \in \mathcal{C}_A$, $|C| \leq k - c_A^{\texttt{col}}$.

▶ **Observation 25 (★).** *Let C be a maximal clique of $G - S$. A vertex $v \in C$ is a b-vertex of some b-coloring of G with k colors that extends $\texttt{col}$, if and only if $|C| = k - c_A^{\texttt{col}}$.*

For each $A \subseteq S$, let $C_A^{\max} \in \mathcal{C}_A$ denote a clique of maximum cardinality among the cliques in $\mathcal{C}_A$. Note that for any $v \in \bigcup \mathcal{C}_A \setminus C_A^{\max}$, by Observation 25, if (G, k) is a Yes-instance of b-Coloring witnessed by a coloring that extends $\texttt{col}$, then there exists a k-b-coloring that extends $\texttt{col}$ where, if v is a b-vertex, then it is not the unique b-vertex of its color, as $C_A^{\max}$ would also contain one such vertex. This is the idea behind the next reduction rule, which deletes vertices until the number of vertices in all the cliques of $\mathcal{C}_A$ is bounded.

▶ **Reduction Rule 26.** *If there exists $A \subseteq S$ such that $|\bigcup \mathcal{C}_A| \geq k - c_A^{\texttt{col}} + 1$, then let $v \in \bigcup \mathcal{C}_A \setminus C_A^{\max}$ and delete v from the graph.*

▶ **Lemma 27 (★).** *Reduction Rule 26 is safe, i.e., under its preconditions, G has a b-coloring with k colors if and only if $G \setminus v$ has a b-coloring with k colors.*

Consider the instance obtained after the exhaustive application of Reduction Rule 26. For brevity of notation let the instance be $(G, k, S, \texttt{col})$ where S is a twin cover of G of size at most t and $\texttt{col}$ is a proper coloring of $G[S]$ with colors from $\{1, \ldots, \min\{k, t\}\}$. Further, for each $A \subseteq S$, the number of vertices present in the union of the cliques in $\mathcal{C}_A$ is at most $k - c_A^{\texttt{col}}$ and therefore $\mathcal{C}_A$ contains at most one clique of size $k - c_A^{\texttt{col}}$. We call such an instance a *cleaned* instance.

▶ **Lemma 28 (★).** *If a cleaned instance $(G, k, S, \texttt{col})$ is a Yes-instance, then there exists a k-b-coloring where for each $A \subseteq S$, the vertices of $\bigcup \mathcal{C}_A$ get distinct colors.*

The safeness of the following reduction rule follows from Lemma 28.

▶ **Reduction Rule 29.** *For each $A \subseteq S$, delete the cliques in $\mathcal{C}_A \setminus \{C_A^{\max}\}$ and add a new clique of size $|\bigcup \mathcal{C}_A \setminus C_A^{\max}|$ whose neighbourhood in G is exactly A.*

▶ **Lemma 30 (★).** *When Reduction Rules 26 and 29 are no longer applicable, the neighbourhood diversity of G is at most $2^{t+1} + t$.*

Proof of Theorem 24. The algorithm starts by finding a twin-cover of G of size t in $\mathcal{O}(1.2378^t + tn + m)$ time [15]. The algorithm guesses the restriction of the k-b-coloring of G onto the vertices of S. Assuming that the colors of S are from the set $\{1, \ldots, \min\{k, t\}\}$ in the k-b-coloring, the number of guesses is at most $t! = 2^{\mathcal{O}(t \log t)}$. Applying Reduction Rule 26 on this instance, we get a cleaned instance. Then applying Reduction Rule 29 on this instance, by Lemma 30 we conclude that the neighbourhood diversity of G is at most $2^{t+1} + t$. Applying all reduction rules can be done in time $\mathcal{O}(2^t \cdot n)$. Using the algorithm of Theorem 22, we solve the problem in $2^{2^{\mathcal{O}(t)}} \cdot \log n + \mathcal{O}(n)$ time. If for neither of the guessed colorings of S, the above algorithm reports Yes, then report a No. Otherwise, report Yes. ◀

6 Conclusion

We explored the landscape of structural parameterizations of b-Coloring. We showed that the problem is XNLP-complete parameterized by pathwidth, which implies it is W[t]-hard for any t by pathwidth, and as a consequence, by treewidth and clique-width as well. Recall that b-Coloring was already known to be XP parameterized by clique-width. The algorithm of [18] runs in time $n^{2^{O(w)}}$, where w is the clique-width of the input graph (which is tight under the Exponential Time Hypothesis). Since graphs of treewidth t have clique-width $2^{\Theta(t)}$ [11], this results in an XP algorithm for b-Coloring parameterized by treewidth with running time $n^{2^{2^{O(t)}}}$. It would be interesting to investigate if this dependence on the treewidth can be improved, and accompanied by a matching lower bound under the ETH.

On the positive side, we showed b-Coloring to be FPT parameterized by neighborhood diversity and twin cover, two generalizations of vertex cover to more dense graphs. A parameter that generalizes both neighborhood diversity and twin cover is modular-width, defined by Gajarský, Lampis and Ordyniak [14]. The complexity of b-Coloring parameterized by modular-width remains an interesting open problem.

References

1 Davi de Andrade and Ana Silva. (Sub)fall coloring and b-coloring parameterized by treewidth. In *Anais do VII Encontro de Teoria da Computação, ETC 2022*, pages 69–72, 2022.

2 Benjamin Bergougnoux, Jan Dreier, and Lars Jaffke. A logic-based algorithmic meta-theorem for mim-width. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 3282–3304. SIAM, 2023. doi:10.1137/1.9781611977554.ch125.

3 Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. In Michael A. Bekos and Michael Kaufmann, editors, *Proceedings of the 48th International Workshop Graph-Theoretic Concepts in Computer Science, WG 2022*, volume 13453 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2022. doi:10.1007/978-3-031-15914-5_7.

4 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima. XNLP-completeness for parameterized problems on graphs with a linear structure. In Holger Dell and Jesper Nederlof, editors, *Proceedings of the 17th International Symposium on Parameterized and Exact Computation, IPEC 2022*, volume 249 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.8.

5 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *Proceedings 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 193–204, 2021. doi:10.1109/FOCS52979.2021.00027.

6 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *Journal of the ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

7 Flavia Bonomo, Guillermo Durán, Frederic Maffray, Javier Marenco, and Mario Valencia-Pabon. On the b-coloring of cographs and P_4-sparse graphs. *Graphs and Combinatorics*, 25(2):153–167, 2009.

8 Flavia Bonomo, Oliver Schaudt, Maya Stein, and Mario Valencia-Pabon. b-Coloring is NP-hard on co-bipartite graphs and polytime solvable on tree-cographs. *Algorithmica*, 73(2):289–305, 2015.

9 Victor A. Campos, Carlos V. Lima, Nicolas A. Martins, Leonardo Sampaio, Marcio C. Santos, and Ana Silva. The b-chromatic index of graphs. *Discrete Mathematics*, 338(11):2072–2079, 2015.

10 Victor A. Campos, Cláudia Linhares-Sales, Rudini Sampaio, and Ana Karolinna Maia. Maximization coloring problems on graphs with few P_4. *Discrete Applied Mathematics*, 164:539–546, 2014.

11 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005. doi:10.1137/S0097539701385351.

12 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.

13 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.

14 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation*, pages 163–176, Cham, 2013. Springer International Publishing.

15 Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In Dániel Marx and Peter Rossmanith, editors, *Parameterized and Exact Computation*, pages 259–271, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

16 Frédéric Havet, Claudia Linhares Sales, and Leonardo Sampaio. b-Coloring of tight graphs. *Discrete Applied Mathematics*, 160(18):2709–2715, 2012.

17 Robert W. Irving and David F. Manlove. The b-chromatic number of a graph. *Discrete Applied Mathematics*, 91(1-3):127–141, 1999.

18 Lars Jaffke, Paloma T. Lima, and Daniel Lokshtanov. b-Coloring parameterized by clique-width. *Theory of Computing Systems*, 2023. To appear. Conference version in STACS 2021, pages 43:1–43:15.

19 Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In Avrim Blum, editor, *Proceedings of the 10th Innovations in Theoretical Computer Science Conference, ITCS 2019*, volume 124 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl, 2019. arxiv:1803.04744. doi:10.4230/LIPIcs.ITCS.2019.43.

20 Martin Koutecký. A note on coloring $(4K_1, C_4, C_6)$-free graphs with a C_7. *Graphs Comb.*, 38(5):149, 2022. doi:10.1007/s00373-022-02553-4.

21 Jan Kratochvíl, Zsolt Tuza, and Margit Voigt. On the b-chromatic number of graphs. In *WG 2002*, pages 310–320, 2002.

22 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.

23 Ana Shirley Ferreira da Silva. *The b-chromatic number of some tree-like graphs*. PhD thesis, Université Joseph-Fourier-Grenoble I, 2010.

24 Clara Inés Betancur Velasquez, Flavia Bonomo, and Ivo Koch. On the b-coloring of P_4-tidy graphs. *Discrete Applied Mathematics*, 159(1):60–68, 2011.

Clustering What Matters in Constrained Settings

Improved Outlier to Outlier-Free Reductions

Ragesh Jaiswal[1]
CSE, IIT Delhi, India

Amit Kumar
CSE, IIT Delhi, India

Abstract

Constrained clustering problems generalize classical clustering formulations, e.g., k-median, k-means, by imposing additional constraints on the feasibility of a clustering. There has been significant recent progress in obtaining approximation algorithms for these problems, both in the metric and the Euclidean settings. However, the outlier version of these problems, where the solution is allowed to leave out m points from the clustering, is not well understood. In this work, we give a general framework for reducing the outlier version of a constrained k-median or k-means problem to the corresponding outlier-free version with only $(1+\varepsilon)$-loss in the approximation ratio. The reduction is obtained by mapping the original instance of the problem to $f(k,m,\varepsilon)$ instances of the outlier-free version, where $f(k,m,\varepsilon) = \left(\frac{k+m}{\varepsilon}\right)^{O(m)}$. As specific applications, we get the following results:

- First FPT (*in the parameters k and m*) $(1+\varepsilon)$-approximation algorithm for the outlier version of capacitated k-median and k-means in Euclidean spaces with *hard* capacities.
- First FPT (*in the parameters k and m*) $(3+\varepsilon)$ and $(9+\varepsilon)$ approximation algorithms for the outlier version of capacitated k-median and k-means, respectively, in general metric spaces with *hard* capacities.
- First FPT (*in the parameters k and m*) $(2-\delta)$-approximation algorithm for the outlier version of the k-median problem under the Ulam metric.

Our work generalizes the results of Bhattacharya et al. and Agrawal et al. to a larger class of constrained clustering problems. Further, our reduction works for arbitrary metric spaces and so can extend clustering algorithms for outlier-free versions in both Euclidean and arbitrary metric spaces.

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

Keywords and phrases clustering, constrained, outlier

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.41

Related Version *Full Version*: `https://arxiv.org/abs/2305.00175`

Funding *Ragesh Jaiswal*: The author acknowledges the support from the SERB, MATRICS grant.

1 Introduction

Center-based clustering problems such as k-median and the k-means are important data processing tasks. Given a metric D on a set of n points $\mathcal{X}$ and a parameter k, the goal here is to partition the set of points into k *clusters*, say $C_1, \ldots, C_k$, and assign the points in each cluster to a corresponding *cluster center*, say $c_1, \ldots, c_k$, respectively, such that the objective $\sum_{i=1}^{k} \sum_{x \in C_i} D(x, c_i)^z$ is minimized. Here z is a parameter which is 1 for k-median and 2 for k-means. The *outlier* version of these problems is specified by another parameter m, where a solution is allowed to leave out up to m points from the clusters. Outlier versions capture settings where the input may contain a few highly erroneous data points. Both the

[1] corresponding author

outlier and the outlier-free versions have been well-studied in the literature with constant factor approximations known for both the k-means and the k-median problem [3, 4, 12]. In addition, fixed-parameter tractable (FPT) $(1+\varepsilon)$-approximation algorithms are known for these problems in the Euclidean setting [26, 19, 8]: the running time of such algorithms is of the form $f(k, m, \varepsilon) \cdot poly(n, d)$, where $f()$ is an exponential function of the parameters k, m, ε and d denotes the dimensionality of the points.

A more recent development in clustering problems has been the notion of *constrained clustering*. A constrained clustering problem specifies additional conditions on a feasible partitioning of the input points into k clusters. For example, the r-gathering problem requires that each cluster in a feasible partitioning must contain at least r data points. Similarly, the well-known *capacitated* clustering problem specifies an upper bound on the size of each cluster. Constrained clustering formulations can also capture various types of *fairness* constraints: each data point has a *label* assigned to it, and we may require upper or lower bounds on the number (or fraction) of points with a certain label in each cluster. Table 1 in the Appendix gives a list of some of these problems. FPT (in the parameter k) constant factor approximation algorithms are known for a large class of these problems (see Table 2 in the Appendix).

It is worth noting that constrained clustering problems are distinct from outlier clustering: the former restricts the set of feasible partitioning of input points, whereas the latter allows us to reduce the set of points that need to be partitioned into clusters. There has not been much progress on constrained clustering problems in the outlier setting (also see [25] for unbounded integrality gap for the natural LP relaxation for the outlier clustering versions). In this work, we bridge this gap between the outlier and the outlier-free versions of constrained clustering problems by giving an *almost approximation-preserving* reduction from the former to the latter. As long as the parameters of interest (i.e., k, m) are small, the reduction works in polynomial time. Using our reduction, an FPT α-approximation algorithm for the outlier-free version of a constrained clustering problem leads to an FPT $(\alpha+\varepsilon)$-approximation algorithm for the outlier version of the same problem. For general metric spaces, this implies the first FPT constant-approximation for outlier versions of several constrained clustering problems; and similarly, we get new FPT $(1+\varepsilon)$-approximation algorithms for several outlier constrained clustering problems –see Table 2 in the Appendix for the precise details.

This kind of FPT approximation preserving reduction in the context of Euclidean k-means was first given by [8] using a sampling-based approach. [20] extended the sampling ideas of [8] to general metric spaces but did not give an approximation-preserving reduction. [2] gave a reduction for general metric spaces using a coreset construction. In this work, we use the sampling-based ideas of [8] to obtain an approximation-preserving reduction from the outlier version to the outlier-free version with improved parameters over [2]. Moreover, our reduction works for most known constrained clustering settings as well.

1.1 Preliminaries

We give a general definition of a constrained clustering problem. For a positive integer k, we shall use $[k]$ to denote the set $\{1, \ldots, k\}$. Let $(\mathcal{X}, D)$ denote the metric space with distance function D. For a point x and a subset S of points, we shall use $D(x, S)$ to denote $\min_{y \in S} D(x, y)$. The set $\mathcal{X}$ contains subsets F and X: here X denotes the set of input points and F is the set of points where a center can be located. An outlier constrained clustering problem is specified by the following parameters and functions:

- k: the number of clusters.
- m: the number of points which can be left out from the clusters.

- a function check: given a partitioning $X_0, X_1, \ldots, X_k$ of X (here X_0 is the set of outliers) and centers $f_1, \ldots, f_k$, each lying in the set F, the function $\mathsf{check}(X_0, X_1, \ldots, X_k, f_1, \ldots, f_k)$ outputs 1 iff this is a feasible clustering. For example, in the r-gathering problem, the $\mathsf{check}(X_0, X_1, \ldots, X_k, f_1, \ldots, f_k)$ outputs 1 iff $|X_i| \geq r$ for each $i \in [k]$. The check function depends only on the cardinality of the sets $X_1, \ldots, X_k$ and the locations $f_1, \ldots, f_k$. This already captures many of the constrained clustering problems. Our framework also applies to the more general labelled version (see details below).
- a cost function cost: given a partitioning $X_0, X_1, \ldots, X_k$ of X and centers $f_1, \ldots, f_k$,

$$\mathsf{cost}(X_0, X_1, \ldots, X_k, f_1, \ldots, f_k) := \sum_{i \in [k]} \sum_{x \in X_i} D^z(x, f_i),$$

  where z is either 1 (the outlier constrained k-median problem) or 2 (the outlier constrained k-means problem).

Given an instance $\mathcal{I} = (X, F, k, m, \mathsf{check}, \mathsf{cost})$ of an outlier constrained clustering problem as above, the goal is to find a partitioning $X_0, X_1, \ldots, X_k$ of X and centers $f_1, \ldots, f_k \in F$ such that $|X_0| \leq m$, $\mathsf{check}(X_0, X_1, \ldots, X_k, f_1, \ldots, f_k)$ is 1 and $\mathsf{cost}(X_0, X_1, \ldots, X_k, f_1, \ldots, f_k)$ is minimized. The outlier-free constrained clustering problem is specified as above, except that the parameter m is 0. For the sake of brevity, we leave out the parameter m and the set X_0 while defining the instance $\mathcal{I}$, and functions check and cost.

We shall also consider a more general class of constrained clustering problems, where each input point is assigned a *label*. In other words, an instance $\mathcal{I}$ of such a problem is specified by a tuple $(X, F, k, m, \sigma, \mathsf{check}, \mathsf{cost})$, where $\sigma : X \to L$ for a finite set L. Note that the check function may depend on the function σ. For example, σ could assign a label "red" or "blue" to each point in X and the check function would require that each cluster X_i should have an equal number of red and blue points. In addition to the locations $f_1, \ldots, f_k$, the $\mathsf{check}(X_1, \ldots, X_k, f_1, \ldots, f_k, \sigma)$ function also depends on $|\sigma^{-1}(l) \cap X_j|$ for each $l \in L, j \in [k]$, i.e., the number of points with a particular label in each of the clusters. Indirectly, this also implies that the check function can impose conditions on the labels of the outliers points. For example, the colorful k-median problem discussed in [2] has the constraint that m_i clients from the label type i should be designated as outliers, given that every client has a unique label. Table 1 in the Appendix gives a description of some of these problems.

We shall use the approximate triangle inequality, which states that for $z \in \{1, 2\}$ and any three points $x_1, x_2, x_3 \in \mathcal{X}$,

$$D^z(x_1, x_3) \leq z \left(D^z(x_1, x_2) + D^z(x_2, x_3) \right). \tag{1}$$

1.2 Our results

Our main result reduces the outlier constrained clustering problem to the outlier-free version. In our reduction, we shall also use approximation algorithms for the (unconstrained) k-median and k-means problems. We assume we have a constant factor approximation algorithm for these problems[2]: let $\mathcal{C}$ denote such an algorithm with running time $T_{\mathcal{C}}(n)$ on an input of size n. Note that $\mathcal{C}$ would be an algorithm for the k-means or the k-median problem depending on whether $z = 1$ or 2 in the definition of the cost function.

[2] Several such constant factor approximation algorithms exist [3, 4, 12].

▶ **Theorem 1** (Main Theorem). *Consider an instance $\mathcal{I} = (X, F, k, m, \mathsf{check}, \mathsf{cost})$ of an outlier constrained clustering problem. Let $\mathcal{A}$ be an α-approximation algorithm for the corresponding outlier-free constrained clustering problem; let $T_{\mathcal{A}}(n)$ be the running time of $\mathcal{A}$ on an input of size n. Given a positive $\varepsilon > 0$, there is an $\alpha(1+\varepsilon)$-approximation algorithm for $\mathcal{I}$ with running time $T_{\mathcal{C}}(n) + q \cdot T_{\mathcal{A}}(n) + O\left(n \cdot (k + \frac{m^{z+1}\log m}{\varepsilon^z})\right) + O\left(qm^2(k+m)^3\right)$, where n is the size of $\mathcal{I}$ and $q = f(k, m, \varepsilon) = \left(\frac{k+m}{\varepsilon}\right)^{O(m)}$, and $z = 1$ or 2 depending on the cost function (i.e., $z = 1$ for k-median objection and $z = 2$ for k-means objective).*

The above theorem implies that as long as there is an FPT or polynomial-time approximation algorithm for the constrained, outlier-free k-median or k-means clustering problem, there is an FPT approximation algorithm (with almost the same approximation ratio) for the corresponding outlier version. We prove this result by creating q instances of the outlier-free version of $\mathcal{I}$ and picking the best solution on these instances using the algorithm $\mathcal{A}$. We also extend the above result to the labelled version:

▶ **Theorem 2** (Main Theorem: labelled version). *Consider an instance $\mathcal{I} = (X, F, k, m, \sigma, \mathsf{check}, \mathsf{cost})$ of an outlier constrained clustering problem with labels on input points. Let $\mathcal{A}$ be an α-approximation algorithm for the corresponding outlier-free constrained clustering problem; let $T_{\mathcal{A}}(n)$ be the running time of $\mathcal{A}$ on an input of size n. Given a positive $\varepsilon > 0$, there is an $\alpha(1+\varepsilon)$-approximation algorithm for $\mathcal{I}$ with running time $T_{\mathcal{C}}(n) + q \cdot T_{\mathcal{A}}(n) + O\left(n \cdot (k + \frac{m^{z+1}\log m}{\varepsilon^z})\right) + O\left(q\ell m^2(k+m)^3\right)$, where n is the size of $\mathcal{I}$, $q = f(k, m, \varepsilon) = \left(\frac{(k+m)\ell}{\varepsilon}\right)^{O(m)}$ with ℓ being the number of distinct labels, and $z = 1$ or 2 depending on the cost function (i.e., $z = 1$ for k-median objection and $z = 2$ for k-means objective).*

The algorithms given in Theorem 1 and Theorem 2 are randomized algorithms that guarantee the stated approximation factor with high probability. The consequences of our results for specific constrained clustering problems are summarized in Table 2 in the Appendix. We give the results of related works [8, 20, 2] in the same table to see the contributions of this work. Our contributions can be divided into two main categories:

1. *Matching the best-known result*: This can be further divided into two categories:
   a. *Matching results of [2]*: [2] gives an outlier to outlier-free reduction. We also give such a reduction using a different technique with better parameters. This means that we match all the results of [2], which includes problems such as the classical k-median/means problems, the Matroid k-median problem, the colorful k-median problem, and k-median in certain special metrics. See rows 2-6 in Table 2 given in the Appendix.
   b. *Matching results of [20]*: [20] gives FPT approximation algorithms for certain constrained problems on which the coreset-based approach of [2] is not known to work. See the last row of Table 2. [20] gives algorithms for outlier and outlier-free versions with the same approximation guarantee. Since the best outlier-free approximation is also from [20], our results currently only match the approximation guarantees of [20]. However, if there is an improvement in any of these problems, our results will immediately beat the known outlier results of [20].
2. *Best known results*: Since our results hold for a larger class of constrained problems than earlier works, there are certain problems for which our results give the best-known FPT approximation algorithm. The list includes capacitated k-median/k-means with hard capacities in general metric and Euclidean spaces. It also includes the k-median problem in the Ulam metric. A recent development in the Ulam k-median problem [11] has broken

the 2-approximation barrier. Our reduction allows us to take this development to the outlier setting as well. The outlier-free results from which our best results are derived using our reduction are given in Table 2 (see rows 7-9) given in the Appendix.

1.3 Comparison with earlier work

As discussed earlier, the idea of a reduction from an outlier clustering problem to the corresponding outlier-free version in the context of the Euclidean k-means problem was suggested by [8] using a D^2-sampling based idea. [20] used the sampling ideas to design approximation algorithms for the outlier versions of various constrained clustering problems. However, the approximation guarantee obtained by [20] was limited to $(3+\varepsilon)$ for a large class of constrained k-median and $(9+\varepsilon)$ for the constrained k-means problems, and it was not clear how to extend these techniques to get improved guarantees. As a result, their techniques could not exploit the recent developments by [14] in the design of $(1+2/e+\varepsilon)$ and $(1+8/e+\varepsilon)$ FPT approximation algorithms for the classical outlier-free k-median and k-means problems respectively in general metric spaces. [2] gave an outlier-to-outlier-free reduction, making it possible to extend the above-mentioned FPT approximation guarantees for the outlier-free setting to the outlier setting.

The reduction of [2] is based on the coreset construction by [13] using uniform sampling. A coreset for a dataset is a weighted set of points such that the clustering of the coreset points with respect to any set of k centers is the same (within a $1\pm\varepsilon$ factor) as that of the original set points. The coreset construction in [13] starts with a set C of centers that give constant factor approximation. They consider $O(\log n)$ "ring" around these centers, uniformly sample points from each of these rings, and set the weight of the sampled points appropriately. The number of sampled points, and hence the size of the coreset, is $\left(\frac{|C|\log n}{\varepsilon}\right)^2$. [2] showed that when starting with $(k+m)$ centers that give a constant approximation to the classical $(k+m)$-median problem, the coreset obtained as above has the following additional property: for any set of k centers, the clustering cost of the original set of points excluding m outliers is same (again, within $1\pm\varepsilon$ factor) as that of the coreset, again allowing for exclusion of a subset of m points from it. This means that by trying out all m subsets from the coreset, we ensure that at least one subset acts as a good outlier set. Since the coreset size is $\left(\frac{(k+m)\log n}{\varepsilon}\right)^2$, the number of outlier-free instances that we construct is $\left(\frac{(k+m)\log n}{\varepsilon}\right)^{O(m)}$. Using $(\log n)^{O(m)} = \max\{m^{O(m)}, n^{O(1)}\}$, this is of the form $f(k,m,\varepsilon)\cdot n^{O(1)}$ for a suitable function f. At this point, we note the first quantitative difference from our result. In our algorithm, we save the $(\log n)^{O(m)}$ factor, which also means that the number of instances does not depend on the problem size n. Further, a coreset-based construction restricts the kind of problems it can be applied to. The coreset property that the cost of original points is the same as that of the weighted cost of coreset points holds when points are assigned to the closest center (*i.e., the entire weight of the coreset goes to the closest center*).[3] This works for the classical unconstrained k-median and k-means problems (as well as the few other settings considered in [2]). However, for several constrained clustering problems, it may not hold that every point is assigned to the closest center. There have been some recent developments [5, 10] in designing coresets for constrained clustering settings. However, they have not been shown to apply to the outlier setting. Another recent work [22] designs coresets for the outlier setting, but like [2], it has limited scope and has not been shown to extend for most constrained settings. Our D^z-sampling-based technique has the advantage that instead of running the outlier-free algorithm on a coreset as in [2], it works directly with the dataset.

[3] The reason is how Haussler's lemma is applied to bound the cost difference.

That is, we run the outlier-free algorithm on the dataset (after removing outlier candidates). This also makes our results helpful in weighted settings (e.g., see [11]) where the outlier-free algorithm is known to work only for unweighted datasets – note a coreset is a weighted set).

Recent independent work. In recent and independent work, [17] design similar approximation preserving reductions for a restricted class of constrained clustering settings, namely capacitated clustering and (α, β)-fair clustering. Further, their results are obtained by extending coreset based ideas of [2].

1.4 Our Techniques

In this section, we give a high-level description of our algorithm. Let $\mathcal{I}$ denote an instance of outlier constrained clustering on a set of points X and $\mathcal{O}$ denote an optimal solution to $\mathcal{I}$. The first observation is that the optimal cost of the outlier-free and unconstrained clustering with $k+m$ centers on X is a lower bound on the cost of $\mathcal{O}$ (Claim 1). [4] Let C denote the set of these $(k+m)$ centers (we can use any constant factor approximation for the unconstrained version to find C). The intuition behind choosing C is that the centers in $\mathcal{O}$ should be close to C.

Now we divide the set of m outliers in $\mathcal{O}$ into two subsets: those which are far from C and the remaining ones close to C ("near" outliers). Our first idea is to randomly sample a subset S of $O(m \log m)$ points from X with sampling probability proportional to distance (or square of distance) from the set C. This sampling ensures that S contains the far outliers with high probability (Claim 2). We can then iterate over all subsets of S to guess the exact subset of far outliers. Handling the near outliers is more challenging and forms the heart of the technical contribution of this paper.

We "assign" each near outlier to its closest point in C – let $X^{\mathsf{opt}}_{N,j}$ be the set of outliers assigned to c_j. By iterating over all choices, we can guess the cardinality t_j of each of the sets $X^{\mathsf{opt}}_{N,j}$. We now set up a suitable minimum cost bipartite b-matching instance which assigns a set of t_j points to each center c_j. Let $\widehat{X}_j$ be the set of points assigned to c_j. Our algorithm uses $\cup_j \widehat{X}_j$ as the set of near outliers. In the analysis, we need to argue that there is a way of matching the points in $X^{\mathsf{opt}}_{N,j}$ to $\widehat{X}_j$ whose total cost (sum of distances or squared distances between matched points) is small (Lemma 4). The hope is that we can go from the optimal set of outliers in $\mathcal{O}$ to the ones in the algorithm and argue that the increase in cost is small. Since we are dealing with constrained clustering, we need to ensure that this process does not change the size of each of the clusters. To achieve this, we need to further modify the matching between the two sets of outliers (Lemma 5). Finally, with this modified matching, we are able to argue that the cost of the solution produced by the algorithm is close to that of the optimal solution. The extension to the labelled version follows along similar lines.

In the remaining paper, we prove our two main results, Theorem 1 and Theorem 2. The main discussion will be for Theorem 1 since Theorem 2 is an extension of Theorem 1 that uses the same proof ideas. In the following sections, we give the details of our algorithm (Section 2) and its analysis (Section 3). In Section 3, we discuss the extension to the labelled version.

2 Algorithm

In this section, we describe the algorithm for the outlier constrained clustering problem. Consider an instance $\mathcal{I} = (X, F, k, m, \mathsf{check}, \mathsf{cost})$ of this problem. Recall that the parameter $z = 1$ or 2 depends on whether the cost function is like the k-median or the k-means objective respectively. In addition, we assume the existence of the following algorithms:

[4] This observation was used in both [8] and [2].

- A constant β-factor algorithm $\mathcal{C}$ for the k-median or the k-means problem (depending on $z = 1$ or $z = 2$ respectively): an instance here is specified by a tuple (X', F', k') only, where X' is the set of input points, F' is the set of potential locations for a center, and k' denotes the number of clusters.
- An algorithm $\mathcal{A}$ for the outlier-free version of this problem. An instance here is given by a tuple $(X', F', k, \mathsf{check}, \mathsf{cost})$ where the check and the cost functions are the same as those in $\mathcal{I}$.
- An algorithm $\mathcal{M}$ for the b-matching problem: an instance of the b-matching problem is specified by a weighted bi-partite graph $G = (L, R = \{v_1, \ldots, v_r\}, E)$, with edge e having weight w_e; and a tuple $(t_1, \ldots, t_r)$, where $t_i, i \in [r]$, are non-negative integers. A solution needs to find a subset of E' of E such each vertex of L is incident with at most one edge of E', and each vertex $v_j \in R$ is incident with *exactly* t_j edges of E'. The goal is to find such a set E' of minimum total weight.

We now define D^z-sampling:

▶ **Definition 3.** *Given sets C and X of points, D^z-sampling from X w.r.t. C samples a point $x \in X$, where the probability of sampling x is proportional to $D^z(x, C)$.*

The algorithm is described in Algorithm 1. It first runs the algorithm $\mathcal{C}$ to obtain a set of $(k + m)$ centers C in line 1.2. In line 1.3, we sample a subset S where each point in S is sampled independently using D^z-sampling w.r.t. C. Given a subset Y, we say that a tuple $\boldsymbol{\tau} = (t_1, \ldots, t_{k+m})$ is *valid* if $t_j \geq 0$ for all $j \in [k + m]$, and $\sum_j t_j + |Y| = m$. For each subset Y of size $\leq m$ of S and for each valid tuple $\boldsymbol{\tau}$, the algorithm constructs a solution $(X_0^{(Y,\boldsymbol{\tau})}, X_1^{(Y,\boldsymbol{\tau})}, \ldots, X_k^{(Y,\boldsymbol{\tau})})$, where $X_0^{(Y,\boldsymbol{\tau})}$ denotes the set of outlier points. This is done by first computing the set $X_0^{(Y,\boldsymbol{\tau})}$, and then using the algorithm $\mathcal{A}$ on the remaining points $X \setminus (X_0^{(Y,\boldsymbol{\tau})} \cup Y)$ (line 1.8). To find the set $X_0^{(Y,\boldsymbol{\tau})}$, we construct an instance $\mathcal{I}^{(Y,\boldsymbol{\tau})}$ of b-matching first (line 1.6). This instance is defined as follows: the bipartite graph has the set of $(k + m)$ centers C on the right side and the set of points X on the left side. The weight of an edge between a vertex $v \in C$ and $w \in X$ is equal to $D^z(v, w)$. For each vertex $v_j \in C$, we require that it is matched to exactly t_j points of X. We run the algorithm $\mathcal{M}$ on this instance of b-matching (line 1.7). We define $X_0^{(Y,\boldsymbol{\tau})}$ as the set of points of X matched by this algorithm. Finally, we output the solution of minimum cost (line 1.10).

Algorithm 1 Algorithm for outlier constrained clustering.

1.1 **Input:** $\mathcal{I} := (X, F, k, m, \mathsf{check}, \mathsf{cost})$
1.2 Execute $\mathcal{C}$ on the instance $\mathcal{I}' := (X, F, k + m)$ to obtain a set C of $k + m$ centers.
1.3 Sample a set S of $\lceil \frac{4\beta m \log m}{\varepsilon} \rceil$ points with replacement, each using D^z-sampling from X w.r.t. C.
1.4 **for** *each subset* $Y \subset S, |Y| \leq m$ **do**
1.5 **for** *each valid tuple* $\boldsymbol{\tau} = (t_1, \ldots, t_{k+m})$ **do**
1.6 Construct the instance $\mathcal{I}^{(Y,\tau)}$
1.7 Run $\mathcal{M}$ on $\mathcal{I}^{(Y,\tau)}$ and let $X_0^{(Y,\boldsymbol{\tau})}$ be the set of matched points in X.
1.8 Run the algorithm $\mathcal{A}$ on the instance $(X \setminus (X_0^{(Y,\boldsymbol{\tau})} \cup Y), F, k, \mathsf{check}, \mathsf{cost})$.
1.9 Let $(X_1^{(Y,\boldsymbol{\tau})}, \ldots, X_k^{(Y,\boldsymbol{\tau})})$ be the clustering produced by $\mathcal{A}$.
1.10 Let $(Y^\star, \boldsymbol{\tau}^\star)$ be the pair for which $\mathsf{cost}(X_1^{(Y,\boldsymbol{\tau})}, \ldots, X_k^{(Y,\boldsymbol{\tau})})$ is minimized.
1.11 **Output** $(X_0^{(Y^\star,\boldsymbol{\tau}^\star)}, X_1^{(Y^\star,\boldsymbol{\tau}^\star)}, \ldots, X_k^{(Y^\star,\boldsymbol{\tau}^\star)})$.

3 Analysis

We now analyze Algorithm 1. We refer to the notation used in this algorithm. Let $\mathcal{I} = (X, F, k, m, \mathsf{check}, \mathsf{cost})$ be the instance of the outlier constrained clustering problem. Let $\mathsf{opt}(\mathcal{I})$ denote the optimal cost of a solution for the instance $\mathcal{I}$. Assume that the algorithm $\mathcal{C}$ for the unconstrained clustering problem (used in line 1.2) is a β-approximation algorithm. We overload notation and use $\mathsf{cost}_{\mathcal{I}'}(C)$ to denote the cost of the solution C for the instance $\mathcal{I}'$. Observe that the quantity $\mathsf{cost}_{\mathcal{I}'}(C)$ can be computed as follows: each point in X is assigned to the closest point in C, and then we compute the total cost (which could be the k-median or the k-means cost based on the value of the parameter z) of this assignment. We first relate $\mathsf{cost}_{\mathcal{I}'}(C)$ to $\mathsf{opt}(\mathcal{I})$.

▷ Claim 1. $\mathsf{cost}_{\mathcal{I}'}(C) \leq \beta \cdot \mathsf{opt}(\mathcal{I})$.

Proof. Let $(X_0, X_1, ..., X_k)$ denote the optimal solution for $\mathcal{I}$, where X_0 denotes the set of m outlier points (without loss of generality, we can assume that the number of outlier points in the optimal solution is exactly m). Let $c_1, \ldots, c_k$ be the centers of the clusters $X_1, \ldots, X_k$ respectively. Consider the solution to $\mathcal{I}'$ consisting of centers $C' := X_0 \cup \{c_1, \ldots, c_k\}$. Clearly, $\mathsf{cost}_{\mathcal{I}'}(C') \leq \mathsf{opt}(\mathcal{I})$ (we have inequality here because the solution $X_1, \ldots, X_k$ may not be a Voronoi partition with respect to $c_1, \ldots, c_k$). Since $\mathcal{C}$ is a β-approximation algorithm, we know that $\mathsf{cost}_{\mathcal{I}'}(C) \leq \beta \cdot \mathsf{cost}_{\mathcal{I}'}(C')$. Combining these two facts implies the desired result. ◁

We now consider an optimal solution for the instance $\mathcal{I}$: let $X_0^{\mathsf{opt}}, X_1^{\mathsf{opt}}, \ldots, X_k^{\mathsf{opt}}$ be the partition of the input points X in this solution, with X_0^{opt} being the set of m outliers. Depending on the distance from C, we divide the set X_0^{opt} into two subsets – X_F^{opt} ("far" points) and X_N^{opt} ("near" points) as follows:

$$X_F^{\mathsf{opt}} := \left\{ x \in X_0^{\mathsf{opt}} | D^z(x, C) \geq \frac{\varepsilon\, \mathsf{cost}_{\mathcal{I}'}(C)}{2\beta m} \right\}, \quad X_N^{\mathsf{opt}} := X \setminus X_F^{\mathsf{opt}}.$$

Recall that we sample a set S of $\frac{4\beta m \log m}{\varepsilon}$ clients using D^z-sampling with respect to center set C (line 1.3 in Algorithm 1). Note that the probability of sampling a point x is given by

$$\frac{D^z(x, C)}{\sum_{x' \in X} D^z(x, C)} = \frac{D^z(x, C)}{\mathsf{cost}_{\mathcal{I}'}(C)}. \tag{2}$$

We first show that S contains all the points in X_F^{opt} with high probability.

▷ Claim 2. $\mathbf{Pr}[X_F^{\mathsf{opt}} \subseteq S] \geq 1 - 1/m$.

Proof. Inequality Equation (2) shows that the probability of sampling a point $x \in X_F^{\mathsf{opt}}$ is $\frac{D^z(x,C)}{\mathsf{cost}_{\mathcal{I}'}(C)} \geq \frac{\varepsilon}{2\beta m}$. So the probability that x is not present in S is at most $\left(1 - \frac{\varepsilon}{2\beta m}\right)^{\frac{4\beta m \log m}{\varepsilon}} \leq \frac{1}{m^2}$. The desired result now follows from union bound. ◁

For the rest of the analysis, we assume that the event in Claim 2 holds. We now note that the sum of the cost of assigning X_N^{opt} to C is at most $\varepsilon \cdot \mathsf{opt}(\mathcal{I})$.

▷ Claim 3. $\sum_{x \in X_N^{\mathsf{opt}}} D^z(x, C) \leq \frac{\varepsilon}{2} \cdot \mathsf{opt}(\mathcal{I})$.

Proof. The claim follows from the following sequence of inequalities:

$$\sum_{x \in X_N^{\mathsf{opt}}} D^z(x, C) < \sum_{x \in X_N^{\mathsf{opt}}} \frac{\varepsilon \, \mathsf{cost}_{\mathcal{I}'}(C)}{2\beta m} \leq \sum_{x \in X_N^{\mathsf{opt}}} \frac{\varepsilon \cdot \mathsf{opt}(\mathcal{I})}{2m} \leq \frac{\varepsilon}{2} \cdot \mathsf{opt}(\mathcal{I}),$$

where the first inequality follows from the definition of X_N^{opt} and the second inequality follows from Claim 1. ◁

For every point in X_N^{opt}, we identify the closest center in $C = \{c_1, \ldots, c_{m+k}\}$ (breaking ties arbitrarily). For each $j \in [k+m]$, let $X_{N,j}^{\mathsf{opt}}$ be the set of points in X_N^{opt} which are closest to c_j. Let $\hat{t}_j$ denote $|X_{N,j}^{\mathsf{opt}}|$. Consider an iteration of line 1.7–1.9 where $Y = X_F^{\mathsf{opt}}, \boldsymbol{\tau} = (\hat{t}_1, \ldots, \hat{t}_{k+m})$. Observe that $\boldsymbol{\tau}$ is valid with respect to Y because $\sum_{j \in [m+k]} |\hat{t}_j| + |Y| = m$. Let $\widehat{X}_1, \ldots, \widehat{X}_{m+k}$ be the set of points assigned to $c_1, \ldots, c_{m+k}$ respectively by the algorithm $\mathcal{M}$. Intuitively, we will like to construct a solution where the set of outliers is given by $\widehat{X} := X_F^{\mathsf{opt}} \cup \widehat{X}_1 \cup \cdots \cup \widehat{X}_{m+k}$. We now show that the set $\widehat{X}$ is "close" to X_0^{opt}, the set of outliers in the optimal solution. In order to do this, we set up a bijection $\mu : X_0^{\mathsf{opt}} \to \widehat{X}$, where μ restricted to X_F^{opt} is identity, and μ restricted to any of the sets $X_{N,j}^{\mathsf{opt}}$ is a bijection from $X_{N,j}^{\mathsf{opt}}$ to $\widehat{X}_j$. Such a function μ is possible because for each $j \in [m+k]$, $|X_{N,j}^{\mathsf{opt}}| = |\widehat{X}_j| = \hat{t}_j$. We now prove this closeness property.

▶ **Lemma 4.** $\sum_{x \in X_0^{opt}} D^z(x, \mu(x)) \leq \varepsilon \cdot z \cdot \mathit{opt}(\mathcal{I})$.

Proof. We first note a useful property of the solution given by the algorithm $\mathcal{M}$. One of the possible solutions for the instance $\mathcal{I}^{(Y,\boldsymbol{\tau})}$ could have been assigning $X_{N,j}^{\mathsf{opt}}$ to the center c_j. Since $\mathcal{M}$ is an optimal algorithm for b-matching, we get

$$\sum_{j \in [k+m]} \sum_{x \in \widehat{X}_j} D^z(x, c_j) \leq \sum_{j \in [k+m]} \sum_{x \in X_{N,j}^{\mathsf{opt}}} D^z(x, c_j) = \sum_{x \in X_N^{\mathsf{opt}}} D^z(x, C) \leq \frac{\varepsilon}{2} \cdot \mathsf{opt}(\mathcal{I}), \tag{3}$$

where the last inequality follows from Claim 3. Now,

$$\begin{aligned} \sum_{x \in X_0^{\mathsf{opt}}} D^z(x, \mu(x)) = \sum_{x \in X_N^{\mathsf{opt}}} D^z(x, \mu(x)) &= \sum_{j \in [k+m]} \sum_{x \in X_{N,j}^{\mathsf{opt}}} D^z(x, \mu(x)) \\ &\overset{(1)}{\leq} z \cdot \sum_{j \in [k+m]} \sum_{x \in X_{N,j}^{\mathsf{opt}}} \left(D^z(x, c_j) + D^z(c_j, \mu(x)) \right), \end{aligned} \tag{4}$$

where the first equality follows from the fact that μ is identity on X_F^{opt}. Since μ is a bijection from $X_{N,j}^{\mathsf{opt}}$ to $\widehat{X}_j$, the above can also be written as

$$z \cdot \sum_{j \in [k+m]} \sum_{x \in X_{N,j}^{\mathsf{opt}}} D^z(x, c_j) + z \cdot \sum_{j \in [k+m]} \sum_{x \in \widehat{X}_j} D^z(x, c_j) \leq z \cdot \varepsilon \, \mathsf{opt}(\mathcal{I}),$$

where the last inequality follows from Claim 3 and (3). This proves the desired result. ◀

The mapping μ described above may have the following undesirable property: there could be a point $x \in X_0^{\mathsf{opt}} \cap \widehat{X}$ such that $\mu(x) \neq x$. This could happen if $x \in X_{N,j}^{\mathsf{opt}}$ and $x \in \widehat{X}_i$ where $i \neq j$. We now show that μ can be modified to another bijection $\widehat{\mu} : X_0^{\mathsf{opt}} \to \widehat{X}$ which is identity on $X_0^{\mathsf{opt}} \cap \widehat{X}$. Note that the mapping $\widehat{\mu}$ is only needed for the analysis of the algorithm.

▶ **Lemma 5.** *There is a bijection $\widehat{\mu} : X_0^{opt} \to \widehat{X}$ such that $\widehat{\mu}(x) = x$ for all $x \in X_0^{opt} \cap \widehat{X}$ and*

$$\sum_{x \in X_0^{opt}} D^z(x, \widehat{\mu}(x)) \leq m^{z-1}\, \varepsilon \cdot z \cdot \mathit{opt}(\mathcal{I}).$$

Proof. We construct a directed graph $H = (V_1, E_1)$ where $V_1 = X_0^{\mathsf{opt}} \cup \widehat{X}$. For every $x \in X_0^{\mathsf{opt}}$, we add the directed arc $(x, \mu(x))$ to E_1. Observe that a self loop in H implies that $\mu(x) = x$. Every vertex in $X_0^{\mathsf{opt}} \setminus \widehat{X}$ has 0 in-degree and out-degree 1; whereas a vertex in $\widehat{X} \setminus X_0^{\mathsf{opt}}$ has in-degree 1 and 0 out-degree. Vertices in $\widehat{X} \cap X_0^{\mathsf{opt}}$ have exactly one incoming and outgoing arc (in case of a self-loop, it counts towards both the in-degree and the out-degree of the corresponding vertex).

The desired bijection $\widehat{\mu}$ is initialized to μ. Let $\mathsf{cost}(\widehat{\mu})$ denote $\sum_{x \in X_0^{\mathsf{opt}}} D^z(x, \widehat{\mu}(x))$; define $\mathsf{cost}(\mu)$ similarly. It is easy to check H is vertex disjoint union of directed cycles and paths. In case of a directed cycle C on more than 1 vertex, it must be the case that each of the vertices in C belong to $\widehat{X} \cap X_0^{\mathsf{opt}}$. In this case, we update $\widehat{\mu}$ be defining $\widehat{\mu}(x) = x$ for each $x \in C$. Clearly this can only decrease $\mathsf{cost}(\widehat{\mu})$. Let $P_1, \ldots, P_l$ be the set of directed paths in H. For each path P_j, we perform the following update: let P_j be a path from a_j to b_j. We know that $a_j \in X^{\mathsf{opt}} \setminus \widehat{X}$, $b_j \in \widehat{X} \setminus X_0^{\mathsf{opt}}$ and each internal vertex of P_j lies in $\widehat{X} \cap X_0^{\mathsf{opt}}$. We update $\widehat{\mu}$ as follows; $\widehat{\mu}(a_j) = b_j$ and $\widehat{\mu}(v) = v$ for each internal vertex v of P_j. The overall increase in $\mathsf{cost}(\widehat{\mu})$ is equal to

$$\sum_{j \in [l]} \left(D^z(a_j, b_j) - \sum_{i=1}^{n_j} D^z(v_j^i, v_j^{i-1}) \right), \tag{5}$$

where $a_j = v_j^0, v_j^1, \ldots, v_j^{n_j} = b_j$ denotes the sequence of vertices in P_j. If $z = 1$, triangle inequality shows that the above quantity is at most 0. In case $z = 2$,

$$D^2(a_j, b_j) \leq n_j \left(\sum_{i=1}^{n_j} D^2(v_j^i, v_j^{i-1}) \right),$$

and so the quantity in (5) is at most $(n_j - 1) \sum_{i=1}^{n_j} D^2(v_j^i, v_j^{i-1})$.

It follows that $\mathsf{cost}(\widehat{\mu}) \leq m^{z-1}\mathsf{cost}(\mu)$. The desired result now follows from Lemma 4. ◀

We run the algorithm $\mathcal{A}$ on the outlier-free constrained clustering instance $\mathcal{I}'' = (X \setminus \widehat{X}, F, k, \mathsf{check}, \mathsf{cost})$ (line 1.8 in Algorithm 1). Let $\mathsf{opt}(\mathcal{I}'')$ be the optimal cost of a solution for this instance. The following key lemma shows that $\mathsf{opt}(\mathcal{I}'')$ is close to $\mathsf{opt}(\mathcal{I})$.

▶ **Lemma 6.** $\mathit{opt}(\mathcal{I}'') \leq (1 + \varepsilon^{\frac{1}{z}}(4m+1)^{z-1})\mathit{opt}(\mathcal{I})$.

Proof. We shall use the solution $(X_0^{\mathsf{opt}}, \ldots, X_k^{\mathsf{opt}})$ to construct a feasible solution for $\mathcal{I}''$. For each $j \in [k]$, let Z_j denote $X_j^{\mathsf{opt}} \cap \widehat{X}$. Let $\widehat{\mu}^{-1}(Z_j)$ denote the pre-image under $\widehat{\mu}$ of Z_j. Since $Z_j \subseteq \widehat{X} \setminus X_0^{\mathsf{opt}}$, $\widehat{\mu}^{-1}(Z_j) \subseteq X_0^{\mathsf{opt}} \setminus \widehat{X}$. For each $j \in [k]$, define $X'_j := (X_j^{\mathsf{opt}} \setminus Z_j) \cup \widehat{\mu}^{-1}(Z_j)$.

▷ Claim 4. $\bigcup_{j=1}^{k} X'_j = X \setminus \widehat{X}$.

Proof. For any $j \in [k]$, we have already argued that $\widehat{\mu}^{-1}(Z_j) \subseteq X_0^{\mathsf{opt}} \setminus \widehat{X} \subseteq X \setminus \widehat{X}$. Clearly, $X_j^{\mathsf{opt}} \setminus Z_j \subseteq X \setminus \widehat{X}$. Therefore $X'_j \subseteq X \setminus \widehat{X}$. Therefore, $\cup_{j \in [k]} X'_j \subseteq X \setminus \widehat{X}$. Since $|X'_j| = |X_j^{\mathsf{opt}}|$,

$$\sum_{j \in [k]} |X'_j| = n - m = |X \setminus \widehat{X}|.$$

This proves the claim. ◁

The above claim implies that $(X'_1, \ldots, X'_k)$ is a partition of $X \setminus \widehat{X}$. Since $|X'_j| = |X_j^{\mathsf{opt}}|$ for all $j \in [k]$ and the function check only depends on the cardinality of the sets in the partition, $(X'_1, \ldots, X'_k)$ is a feasible partition (under check) of $X \setminus \widehat{X}$. In the optimal solution for $\mathcal{I}$, let $f_1^{\mathsf{opt}}, \ldots, f_k^{\mathsf{opt}}$ be the k centers corresponding to the clusters $X_1^{\mathsf{opt}}, \ldots, X_k^{\mathsf{opt}}$ respectively. Now,

$$\mathsf{opt}(\mathcal{I}'') \leq \mathsf{cost}(X'_1, \ldots, X'_k) \leq \sum_{j \in [k]} \sum_{x \in X'_j} D^z(x, f_j^{\mathsf{opt}}) \tag{6}$$

For each $j \in [k]$, we estimate the quantity $\sum_{x \in X'_j} D^z(x, f_j^{\mathsf{opt}})$. Using the definition of X'_j and triangle inequality, this quantity can be expressed as

$$\begin{aligned} &\sum_{x \in X_j^{\mathsf{opt}} \setminus Z_j} D^z(x, f_j^{\mathsf{opt}}) + \sum_{x \in \widehat{\mu}^{-1}(Z_j)} D^z(x, f_j^{\mathsf{opt}}) \\ &\qquad \leq \sum_{x \in X_j^{\mathsf{opt}} \setminus Z_j} D^z(x, f_j^{\mathsf{opt}}) + \sum_{x \in \widehat{\mu}^{-1}(Z_j)} \left(D(x, \widehat{\mu}(x)) + D(\widehat{\mu}(x), f_j^{\mathsf{opt}})\right)^z \end{aligned} \tag{7}$$

When $z = 1$, the above is at most (replacing x by $\widehat{\mu}(x)$ in the second expression on RHS)

$$\sum_{x \in X_j^{\mathsf{opt}}} D(x, f_j^{\mathsf{opt}}) + \sum_{x \in Z_j} D(x, \widehat{\mu}(x)).$$

Using this bound in (6), we see that

$$\mathsf{opt}(\mathcal{I}'') \leq \mathsf{opt}(\mathcal{I}) + \sum_{x \in X_0^{\mathsf{opt}}} D(x, \widehat{\mu}(x)) \leq (1 + \varepsilon)\mathsf{opt}(\mathcal{I}),$$

where the last inequality follows from Lemma 5. This proves the desired result for $z = 1$. When $z = 2$, we use the fact that for any two reals a, b,

$$(a + b)^2 \leq (1 + \sqrt{\varepsilon})a^2 + b^2 \left(1 + \frac{1}{\sqrt{\varepsilon}}\right).$$

Using this fact, the expression in the RHS of (7) can be upper bounded by

$$(1 + \sqrt{\varepsilon}) \sum_{x \in X_j^{\mathsf{opt}}} D^2(x, f_j^{\mathsf{opt}}) + \left(1 + \frac{1}{\sqrt{\varepsilon}}\right) \sum_{x \in Z_j} D^2(x, \widehat{\mu}(x)).$$

Substituting this expression in (6) and using Lemma 5, we see that

$$\mathsf{opt}(\mathcal{I}'') \leq (1 + \sqrt{\varepsilon})\mathsf{opt}(\mathcal{I}) + 4m\sqrt{\varepsilon}\mathsf{opt}(\mathcal{I}).$$

This proves the desired result for $z = 2$. ◀

The approximation preserving properties of Theorem 1 follow from the above analysis. For the k-means problem, since the approximation term is $(1 + \sqrt{\varepsilon}(4m + 1))$, we can replace ε with $\varepsilon^2/(4m + 1)^2$ in the algorithm and analysis to obtain a $(1 + \varepsilon)$ factor. Let us quickly check the running time of the algorithm. The algorithm first runs $\mathcal{C}$ that takes $T_{\mathcal{C}}(n)$ time. This is followed by D^z-sampling $O(\frac{m^{z+1} \log m}{\varepsilon^z})$ points, which takes $O(n \cdot (k + \frac{m^{z+1} \log m}{\varepsilon^z}))$ time. The number of iterations of the for-loops is determined by the number of subsets of S, which is $\sum_{i=0}^{m} \binom{|S|}{i} = \left(\frac{m}{\varepsilon}\right)^{O(m)}$, and the number of possibilities for τ, which is at most $\binom{2m+k-1}{m} = (m + k)^{O(m)}$. This gives the number of iterations $q = f(k, m, \varepsilon) = \left(\frac{k+m}{\varepsilon}\right)^{O(m)}$. In every iteration, in addition to running $\mathcal{A}$, we solve a weighted b-matching problem on a bipartite graph $(L \cup R, E)$ where R has $(k + m)$ vertices (corresponding to the $k + m$ centers in the center set C) and L has at most $(k + m) \cdot m$ vertices (considering m closest clients for every center is sufficient which can be found using a pre-processing step). So, every iteration costs $T_{\mathcal{A}}(n) + O((k + m)^3 m^2)$ time. This gives the running time expression in Theorem 1.

Extension to labelled version

In this section, we extend Algorithm 1 to the setting where points in X have labels from a finite set L and the check() function can also depend on the number of points with a certain label in a cluster. The overall structure of Algorithm 1 remains unchanged; we just indicate the changes needed in this algorithm.

Given a non-negative integer p, a label partition of p is defined as a tuple $\psi = (q_1, \ldots, q_{|L|})$ such that $\sum_i q_i = p$. The intuition is that given a set S of size p, q_1 points get the first label in L, q_2 points in S get the second label in L, and so on. Now, given a subset Y, define a valid tuple $\boldsymbol{\tau}$ w.r.t. Y as a tuple $((t_1, \psi_1), \ldots, (t_{k+m}, \psi_{k+m}))$, where (i) $\sum_j t_j + |Y| = m$, and (ii) ψ_j is a label partition of t_j. As in line 1.5 in Algorithm 1, we iterate over all such valid tuples. The definition of a solution to the b-matching instance $\mathcal{I}^{(Y,\tau)}$ changes as follows. Let $\psi_j = (n_j^1, \ldots, n_j^\ell)$, where $\ell = |L|$. Then a solution to $\mathcal{I}^{(Y,\tau)}$ needs to satisfy the condition that for each point $c_j \in C$ and each label $l \in L$, exactly n_j^l points in X are matched to c_j. Note that this also implies that exactly t_j points are matched to c_j. This matching problem can be easily reduced to weighted bipartite matching by making t_j copies of each point c_j, and for each label l, adding edges between n_j^l distinct copies of c_j to vertices of label l only. The rest of the details of Algorithm 1 remain unchanged. Note that the running time of the algorithm changes because we now have to iterate over all partitions of each of the numbers t_j.

The analysis of the algorithm proceeds in an analogous manner as that of Algorithm 1. We just need to consider the iteration of the algorithm, where we correctly guess the size of each of the sets $X_{N,j}^{\mathsf{opt}}$ and the number of points of each label in this set.

References

1 Gagan Aggarwal, Rina Panigrahy, Tomás Feder, Dilys Thomas, Krishnaram Kenthapadi, Samir Khuller, and An Zhu. Achieving anonymity via clustering. *ACM Trans. Algorithms*, 6(3), July 2010. doi:10.1145/1798596.1798602.

2 Akanksha Agrawal, Tanmay Inamdar, Saket Saurabh, and Jie Xue. Clustering what matters: Optimal approximation for clustering with outliers, 2023. arXiv:2212.00696.

3 S. Ahmadian, A. Norouzi-Fard, O. Svensson, and J. Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72, October 2017. doi:10.1109/FOCS.2017.15.

4 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004. doi:10.1137/S0097539702416402.

5 Sayan Bandyapadhyay, Fedor V. Fomin, and Kirill Simonov. On Coresets for Fair Clustering in Metric and Euclidean Spaces and Their Applications. 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 23:1–23:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.23.

6 Suman Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/fc192b0c0d270dbf41870a63a8c76c2f-Paper.pdf.

7 Ioana O. Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R. Schmidt, and Melanie Schmidt. On the Cost of Essentially Fair Clusterings. 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 18:1–18:22, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.18.

8 Anup Bhattacharya, Dishant Goyal, Ragesh Jaiswal, and Amit Kumar. On Sampling Based Algorithms for k-Means. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*, volume 182 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2020.13.

9 Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Faster algorithms for the constrained k-means problem. *Theor. Comp. Sys.*, 62(1):93–115, January 2018. doi:10.1007/s00224-017-9820-7.

10 V. Braverman, V. Cohen-Addad, H. Jiang, R. Krauthgamer, C. Schwiegelshohn, M. Toftrup, and X. Wu. The power of uniform sampling for coresets. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 462–473, Los Alamitos, CA, USA, November 2022. IEEE Computer Society. doi:10.1109/FOCS54457.2022.00051.

11 Diptarka Chakraborty, Debarati Das, and Robert Krauthgamer. Clustering permutations: New techniques with streaming applications. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 31:1–31:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ITCS.2023.31.

12 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002. doi:10.1006/jcss.2002.1882.

13 Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009. doi:10.1137/070699007.

14 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT Approximations for k-Median and k-Means. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.42.

15 Vincent Cohen-Addad and Jason Li. On the Fixed-Parameter Tractability of Capacitated Clustering. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41:1–41:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.41.

16 Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coreset framework for clustering. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 169–182, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451022.

17 Rajni Dabas, Neelima Gupta, and Tanmay Inamdar. Fpt approximations for capacitated/fair clustering with outliers, 2023. arXiv:2305.01471.

18 Hu Ding. Faster balanced clusterings in high dimension. *Theoretical Computer Science*, 842:28–40, 2020. doi:10.1016/j.tcs.2020.07.022.

19 Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In *Proceedings of the twenty-third annual symposium on Computational geometry*, SCG '07, pages 11–18, New York, NY, USA, 2007. ACM. doi:10.1145/1247069.1247072.

20 Dishant Goyal, Ragesh Jaiswal, and Amit Kumar. FPT Approximation for Constrained Metric k-Median/Means. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2020.14.

21 Mohammadtaghi Hajiaghayi, Wei Hu, Jian Li, Shi Li, and Barna Saha. A constant factor approximation algorithm for fault-tolerant k-median. *ACM Trans. Algorithms*, 12(3), April 2016. doi:10.1145/2854153.

22 Lingxiao Huang, Shaofeng H. C. Jiang, Jianing Lou, and Xuan Wu. Near-optimal coresets for robust clustering, 2022. arXiv:2210.10394.

23 Tanmay Inamdar and Kasturi Varadarajan. Fault tolerant clustering with outliers. In Evripidis Bampis and Nicole Megow, editors, *Approximation and Online Algorithms*, pages 188–201, Cham, 2020. Springer International Publishing.

24 Ravishankar Krishnaswamy, Amit Kumar, Viswanath Nagarajan, Yogish Sabharwal, and Barna Saha. The matroid median problem. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 1117–1130, USA, 2011. Society for Industrial and Applied Mathematics.

25 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 646–659, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188882.

26 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, February 2010. doi:10.1145/1667053.1667054.

27 Clemens Rösner and Melanie Schmidt. Privacy Preserving Clustering with Constraints. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 96:1–96:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.96.

A Tables

Table 1 The table defines various outlier-free versions of the constrained k-median problem. The k-means versions are defined similarly using D^2 instead of D. We include a few references. The problems are categorized based on the *type* of constraints. There are three main types of constraints (i) *size* (constraints on the cluster size), (ii) *center* (constraints on the points a center can service), and (iii) *label* (constraints on the label of points in clusters). A constrained problem can have a combination of these constraint types.

Problem	Description
Unconstrained k-median (*Constraint type: unconstrained*)	*Input*: (F, X, k) *Output*: $(X_1, ..., X_k, f_1, ..., f_k)$ *Constraints*: None, i.e., $\mathsf{check}(X_1, ..., X_k, f_1, ..., f_k)$ always equals 1. *Objective*: Minimise $\sum_i \sum_{x \in X_i} D(x, f_i)$. (This includes various versions corresponding to specific metrics such as Ulam metric on permutations, metric spaces with constant doubling dimension etc.)
Fault-tolerant k-median (*Constraint type: unconstrained but labelled*) [21, 23]	*Input*: (F, X, k) and a number $h(x) \leq k$ for every facility $x \in X$ *Output*: $(f_1, ..., f_k)$ *Constraints*: None. *Objective*: Minimise $\sum_{x \in X} \sum_{j=1}^{h(x)} D(x, f_{\pi_x(j)})$, where $\pi_x(j)$ is the index of j^{th} nearest center to x in $(f_1, ..., f_k)$ (*Label*: $h(x)$ may be regarded as the label of the client x. So, the number of distinct labels $\ell \leq k$.)
Balanced k-median (*Constraint type: size*) [1, 18]	*Input*: (F, X, k) and integers $(r_1, ..., r_k)$, $(l_1, ..., l_k)$, *Output*: $(X_1, ..., X_k, f_1, ..., f_k)$ *Constraints*: X_i should have at least r_i and at most l_i clients, i.e., $\mathsf{check}(X_1, ..., X_k, f_1, ..., f_k) = 1$ iff $\forall i, r_i \leq \lvert X_i \rvert \leq l_i$. *Objective*: Minimise $\sum_i \sum_{x \in X_i} D(x, f_i)$. (Versions corresponding to specific values of r_i's and l_i's are known by different names. The version corresponding to $l_1 = ... = l_k = \lvert X \rvert$ is called the r-gather problem and the version where $r_1 = ... = r_k = 0$ is called the l-capacity problem.)
Capacitated k-median (*Constraint type: center + size*) [15]	*Input*: (F, X, k) and with capacity $s(f)$ for every facility $f \in F$ *Output*: $(X_1, ..., X_k, f_1, ..., f_k)$ *Constraints*: The number of clients, X_i, assigned to f_i is at most $s(f_i)$, i.e., $\mathsf{check}(X_1, ..., X_k, f_1, ..., f_k) = 1$ iff $\forall i, \lvert X_i \rvert \leq s(f_i)$. *Objective*: Minimise $\sum_i \sum_{x \in X_i} D(x, f_i)$.
Matroid k-median (*Constraint type: center*) [24, 14]	*Input*: (F, X, k) and a Matroid on F *Output*: $(X_1, ..., X_k, f_1, ..., f_k)$ *Constraints*: The number of clients, X_i, assigned to f_i is at most $s(f_i)$, i.e., $\mathsf{check}(X_1, ..., X_k, f_1, ..., f_k) = 1$ iff $\{f_1, ..., f_k\}$ is an independent set of the Matroid . *Objective*: Minimise $\sum_i \sum_{x \in X_i} D(x, f_i)$.
Strongly private k-median (*Constraint type: label + size*) [27]	*Input*: (F, X, k) and numbers $(l_1, ..., l_w)$. Each client has a label $\in \{1, ..., w\}$. *Output*: $(X_1, ..., X_k, f_1, ..., f_k)$ *Constraints*: Every X_i has at least l_j clients with label j, i.e., $\mathsf{check}(X_1, ..., X_k, f_1, ..., f_k) = 1$ iff $\forall i, j, \lvert X_i \cap S_j \rvert \geq l_j$, where S_j is the set of clients with label j . *Objective*: Minimise $\sum_i \sum_{x \in X_i} D(x, f_i)$. (*Labels*: The number of distinct labels $\ell = w$).
l-diversity k-median (*Constraint type: label + size*) [7]	*Input*: (F, X, k) and a number $l > 1$. Each client has one colour from $\in \{1, ..., w\}$ *Output*: $(X_1, ..., X_k, f_1, ..., f_k)$ *Constraints*: The fraction of clients with colour j in every X_i is at least $1/l$, i.e., $\mathsf{check}(X_1, ..., X_k, f_1, ..., f_k) = 1$ iff $\forall i, j, \lvert X_i \cap S_j \rvert \leq \lvert X_i \rvert / l$, where S_j is the set of clients with colour j . *Objective*: Minimise $\sum_i \sum_{x \in X_i} D(x, f_i)$. (*Labels*: Each colour can be regarded as a label and hence the number of distinct labels $\ell = w$).
Fair k-median (*Constraint type: label + size*) [7, 6]	*Input*: (F, X, k) and fairness values $(\alpha_1, ..., \alpha_w), (\beta_1, ..., \beta_w)$. Each client has colours from $\in \{1, ..., w\}$ *Output*: $(X_1, ..., X_k, f_1, ..., f_k)$ *Constraints*: The fraction of clients with colour j in every X_i is between α_j and β_j, i.e., $\mathsf{check}(X_1, ..., X_k, f_1, ..., f_k) = 1$ iff $\forall i, j, \alpha_j \lvert X_i \rvert \leq \lvert X_i \cap S_j \rvert \leq \beta \lvert X_i \rvert$, where S_j is the set of clients with colour j . *Objective*: Minimise $\sum_i \sum_{x \in X_i} D(x, f_i)$. (There are two versions: (i) each client has a unique label, and (ii) a client can have multiple labels.) (*Labels*: For the first version $\ell = w$ and for the second version $\ell = 2^w$.)

Table 2 A $\times$ means that the techniques are not known to apply to the problem. The new results that do not follow from the previously known results are shaded. The results that were not explicitly reported but follow from the techniques in the paper are shaded. The techniques of [2] do not apply to the Ulam k-median problem since the outlier-free algorithm works on unweighted instances. Note that all the FPT $(3+\varepsilon)$ and $(9+\varepsilon)$ approximations for the outlier-free versions (leftmost column) in the last row follow from the outlier-free results in [20]. However, the approximation guarantees in the rightmost column depend on those in the leftmost. This means, unlike the rigid $(3+\varepsilon)$ and $(9+\varepsilon)$ approximation of [20] in the middle column, the approximation guarantee in the rightmost column will improve with every improvement in the leftmost.

Problem	**Outlier-free**	**Outlier version**		
		[20]	[2]	**This work**
Euclidean k-means (i.e., $F=\mathbb{R}^d, X\subset\mathbb{R}^d$)	$(1+\varepsilon)$ [9]	$\times$	$(1+\varepsilon)$	$(1+\varepsilon)$
k-median	$\left(1+\frac{2}{e}+\varepsilon\right)$ [14]	$(3+\varepsilon)$	$\left(1+\frac{2}{e}+\varepsilon\right)$	$\left(1+\frac{2}{e}+\varepsilon\right)$
k-means	$\left(1+\frac{8}{e}+\varepsilon\right)$ [14]	$(9+\varepsilon)$	$\left(1+\frac{8}{e}+\varepsilon\right)$	$\left(1+\frac{8}{e}+\varepsilon\right)$
k-median/means in metrics: (i) constant doubling dimension (ii) metrics induced by graphs of bounded treewidth (iii) metrics induced by graphs that exclude a fixed graph as a minor	$(1+\varepsilon)$ [16]	$(3+\varepsilon)$ k-median $(9+\varepsilon)$ k-means	$(1+\varepsilon)$	$(1+\varepsilon)$
Matroid k-median	$(2+\varepsilon)$ [14]	$(3+\varepsilon)$	$(2+\varepsilon)$	$(2+\varepsilon)$
Colourful k-median	$\left(1+\frac{2}{e}+\varepsilon\right)$ [14]	$(3+\varepsilon)$	$\left(1+\frac{2}{e}+\varepsilon\right)$	$\left(1+\frac{2}{e}+\varepsilon\right)$
Ulam k-median (here $F=X$)	$(2-\delta)$ [11]	$(2+\varepsilon)$	$\times$	$(2-\delta)$
Euclidean Capacitated k-median/means	$(1+\varepsilon)$ [15]	$\times$	$\times$	$(1+\varepsilon)$
Capacitated k-median Capacitated k-means	$(3+\varepsilon)$ $(9+\varepsilon)$ [15]	$\times$ $\times$	$\times$ $\times$	$(3+\varepsilon)$ $(9+\varepsilon)$
Uniform/non-uniform r-gather k-median/means (uniform implies $r_1=r_2=\ldots=r_k$) Uniform/non-uniform l-capacity k-median/means (uniform implies $l_1=l_2=\ldots=l_k$) Uniform/non-uniform balanced k-median/means (uniform implies $r_1=r_2=\ldots=r_k$ and $l_1=l_2=\ldots=l_k$) Uniform/non-uniform fault tolerant k-median/means (uniform implies same $h(x)$ for every x) Strongly private k-median/means l-diversity k-median/means Fair k-median/means	$(3+\varepsilon)$ (k-median) $(9+\varepsilon)$ (k-means) [20]	$(3+\varepsilon)$ (k-median) $(9+\varepsilon)$ (k-means)	$\times$ $\times$	$(3+\varepsilon)$ (k-median) $(9+\varepsilon)$ (k-means)

Single-Exponential FPT Algorithms for Enumerating Secluded $\mathcal{F}$-Free Subgraphs and Deleting to Scattered Graph Classes

Bart M. P. Jansen
Eindhoven University of Technology, The Netherlands

Jari J. H. de Kroon
Eindhoven University of Technology, The Netherlands

Michał Włodarczyk
University of Warsaw, Poland

Abstract

The celebrated notion of important separators bounds the number of small (S, T)-separators in a graph which are "farthest from S" in a technical sense. In this paper, we introduce a generalization of this powerful algorithmic primitive, tailored to undirected graphs, that is phrased in terms of k-*secluded* vertex sets: sets with an open neighborhood of size at most k.

In this terminology, the bound on important separators says that there are at most 4^k maximal k-secluded connected vertex sets C containing S but disjoint from T. We generalize this statement significantly: even when we demand that $G[C]$ avoids a finite set $\mathcal{F}$ of forbidden induced subgraphs, the number of such maximal subgraphs is $2^{\mathcal{O}(k)}$ and they can be enumerated efficiently. This enumeration algorithm allows us to make significant improvements for two problems from the literature.

Our first application concerns the Connected k-Secluded $\mathcal{F}$-free subgraph problem, where $\mathcal{F}$ is a finite set of forbidden induced subgraphs. Given a graph in which each vertex has a positive integer weight, the problem asks to find a maximum-weight connected k-secluded vertex set $C \subseteq V(G)$ such that $G[C]$ does not contain an induced subgraph isomorphic to any $F \in \mathcal{F}$. The parameterization by k is known to be solvable in triple-exponential time via the technique of recursive understanding, which we improve to single-exponential.

Our second application concerns the deletion problem to *scattered graph classes*. A scattered graph class is defined by demanding that every connected component is contained in at least one of the prescribed graph classes $\Pi_1, \ldots, \Pi_d$. The deletion problem to a scattered graph class is to find a vertex set of size at most k whose removal yields a graph from the class. We obtain a single-exponential algorithm whenever each class Π_i is characterized by a finite number of forbidden induced subgraphs. This generalizes and improves upon earlier results in the literature.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases fixed-parameter tractability, important separators, secluded subgraphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.42

Related Version *Full Version*: `https://arxiv.org/abs/2309.11366` [24]

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).

1 Introduction

Graph separations have played a central role in algorithmics since the discovery of min-cut/max-flow duality and the polynomial-time algorithm to compute a maximum flow [15]. Nowadays, more complex separation properties are crucial in the study of parameterized complexity, where the goal is to design algorithms for NP-hard problems whose running time can be bounded as $f(k) \cdot n^{\mathcal{O}(1)}$ for some function f that depends only on the *parameter* k of the input. There are numerous graph problems which either explicitly involve finding separations of a certain kind (such as Multiway Cut [33], Multicut [4, 36], k-Way Cut [25], and Minimum Bisection [11]) or in which separation techniques turn out to be instrumental for an efficient solution (such as Directed Feedback Vertex Set [7] and Almost 2-SAT [39]).

The field of parameterized complexity has developed a robust toolbox of techniques based on graph separators, e.g., treewidth reduction [35], important separators [34], shadow removal [36], discrete relaxations [12, 18, 19, 20], protrusion replacement [37], randomized contractions and recursive understanding [8, 10, 31], and flow augmentation [26, 27]. These powerful techniques allowed a large variety of graph separation problems to be classified as fixed-parameter tractable. However, this power comes at a cost. The running times for many applications of these techniques are superexponential: of the form $2^{p(k)} \cdot n^{\mathcal{O}(1)}$ for a high-degree polynomial p, double-exponential, or even worse. Discrete relaxations form a notable exception, which we discuss in Section 4.

The new algorithmic primitive we develop can be seen as an extension of important separators [34] [9, §8]. The study of important separators was pioneered by Marx [33, 34] and refined by follow-up work by several authors [6, 29], which was recognized by the EATCS-IPEC Nerode Prize 2020 [3]. The technique is used to bound the number of extremal (S, T)-separators in an n-vertex graph G with vertex sets S and T. The main idea is that, even though the number of distinct inclusion-minimal (S, T)-separators (which are vertex sets potentially intersecting $S \cup T$) of size at most k can be as large as $n^{\Omega(k)}$, the number of *important* separators which leave a maximal vertex set reachable from S, is bounded by 4^k. For Multiway Cut, a pushing lemma [33, Lem. 6] shows that there is always an optimal solution that contains an important separator, which leads to an algorithm solving the problem in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. Important separators also form a key ingredient for solving many other problems such as Multicut [4, 36] and Directed Feedback Vertex Set [7].

For our purposes, it will be convenient to view the bound on the number of important separators through the lens of *secluded subgraphs*.

▶ **Definition 1.** *A vertex set $S \subseteq V(G)$ or induced subgraph $G[S]$ of an undirected graph G is said to be* k-secluded *if $|N_G(S)| \leq k$, that is, the number of vertices outside S which are adjacent to a vertex of S is bounded by k.*

A vertex set S in a graph G is called seclusion-maximal *with respect to a certain property Π if S satisfies Π and for all sets $S' \supsetneq S$ that satisfy Π we have $|N_G(S')| > |N_G(S)|$.*

Hence a seclusion-maximal set with property Π is inclusion-maximal among all subsets with the same size neighborhood. Consequently, the number of inclusion-maximal k-secluded sets satisfying Π is at most the number of seclusion-maximal k-secluded sets with that property.

Using the terminology of seclusion-maximal subgraphs, the bound on the number of important (S, T)-separators of size at most k in a graph G is equivalent to the following statement: in the graph G' obtained from G by inserting a new source r adjacent to S, the number of *seclusion-maximal* k-secluded connected subgraphs C containing r but no vertex of T is bounded by 4^k. The neighborhoods of such subgraphs C correspond exactly to the important (S, T)-separators in G.

While a number of previously studied cut problems [30, 35] place further restrictions on the vertex set that forms the separator (for example, requiring it to induce a connected graph or independent set) our generalization instead targets the structure of the k-secluded connected subgraph C. We will show that, for any fixed finite family $\mathcal{F}$ of graphs, the number of k-secluded connected subgraphs C as above which are seclusion-maximal with respect to satisfying the additional requirement that $G[C]$ contains no induced subgraph isomorphic to a member of $\mathcal{F}$ is still bounded by $2^{\mathcal{O}(k)}$. Observe that the case $\mathcal{F} = \emptyset$ corresponds to the original setting of important separators. Note that a priori, it is not even clear that the number of seclusion-maximal graphs of this form can be bounded by any function $f(k)$, let alone a single-exponential one.

Our contribution

Having introduced the background of secluded subgraphs, we continue by stating our result exactly. This will be followed by a discussion on its applications.

For a finite set $\mathcal{F}$ of graphs we define $||\mathcal{F}|| := \max_{F \in \mathcal{F}} |V(F)|$, the maximum order of any graph in $\mathcal{F}$. We say that a graph is $\mathcal{F}$-free if it does not contain an *induced* subgraph isomorphic to a graph in $\mathcal{F}$. Our generalization of important separators is captured by the following theorem, in which we use $\mathcal{O}_{\mathcal{F}}(\ldots)$ to indicate that the hidden constant depends on $\mathcal{F}$.

▶ **Theorem 2.** *Let $\mathcal{F}$ be a finite set of graphs. For any n-vertex graph G, non-empty vertex set $S \subseteq V(G)$, potentially empty $T \subseteq V(G) \setminus S$, and integer k, the number of k-secluded induced subgraphs $G[C]$ which are seclusion-maximal with respect to being connected, $\mathcal{F}$-free, and satisfying $S \subseteq C \subseteq V(G) \setminus T$, is bounded by $2^{\mathcal{O}_{\mathcal{F}}(k)}$. A superset of size $2^{\mathcal{O}_{\mathcal{F}}(k)}$ of these subgraphs can be enumerated in time $2^{\mathcal{O}_{\mathcal{F}}(k)} \cdot n^{||\mathcal{F}||+\mathcal{O}(1)}$ and polynomial space.*

The single-exponential bound given by the theorem is best-possible in several ways. Existing lower bounds on the number of important separators [9, Fig. 8.5] imply that even when $\mathcal{F} = \emptyset$ the bound cannot be improved to $2^{o(k)}$. The term $n^{||\mathcal{F}||}$ in the running time is unlikely to be avoidable, since even testing whether a single graph is $\mathcal{F}$-free is equivalent to Induced Subgraph Isomorphism and cannot be done in time $n^{o(||\mathcal{F}||)}$ [9, Thm. 14.21] assuming the Exponential Time Hypothesis (ETH) due to lower bounds for k-Clique.

The polynomial space bound applies to the internal space usage of the algorithm, as the output size may be exponential in k. More precisely, we consider polynomial-space algorithms equipped with a command that outputs an element and we require that for each element in the enumerated set, this command is called at least once. The algorithm could also enumerate just the set in question (rather than its superset) by postprocessing the output and comparing each pair of enumerated subgraphs. However, storing the entire output requires exponential space.

By executing the enumeration algorithm for every singleton set S of the form $\{v\}$, $v \in V(G)$, and $T = \emptyset$, we immediately obtain the following.

▶ **Corollary 3.** *Let $\mathcal{F}$ be a finite set of graphs. For any n-vertex graph G and integer k, the number of k-secluded induced subgraphs $G[C]$ which are seclusion-maximal with respect to being connected and $\mathcal{F}$-free is $2^{\mathcal{O}_{\mathcal{F}}(k)} \cdot n$. A superset of size $2^{\mathcal{O}_{\mathcal{F}}(k)} \cdot n$ of these subgraphs can be enumerated in time $2^{\mathcal{O}_{\mathcal{F}}(k)} \cdot n^{||\mathcal{F}||+\mathcal{O}(1)}$ and polynomial space.*

Note that we require that the set $\mathcal{F}$ of forbidden induced subgraphs is finite. This is necessary in order to obtain a bound independent of n in Theorem 2. For example, the number of seclusion-maximal ($k = 1$)-secluded connected subgraphs C containing a prescribed vertex r for which C induces an acyclic graph is already as large as $n-1$ in a graph consisting

of a single cycle, since each way of omitting a vertex other than r gives such a subgraph. For this case, the forbidden induced subgraph characterization $\mathcal{F}$ consists of all cycles. Extending this example to a flower structure of k cycles of length n/k pairwise intersecting only in r shows that the number of seclusion-maximal k-secluded $\mathcal{F}$-free connected subgraphs containing r is $\Omega(n^k/k^k)$ and cannot be bounded by $f(k) \cdot n^{\mathcal{O}(1)}$ for any function f.

We give two applications of Theorem 2 to improve the running time of existing super-exponential (or even triple-exponential) parameterized algorithms to single-exponential, which is optimal under ETH. For each application, we start by presenting some context.

Application I: Optimization over connected k-secluded $\mathcal{F}$-free subgraphs

The computation of secluded versions of graph-theoretic objects such as paths [2, 5, 32], trees [13], Steiner trees [14], or feedback vertex sets [1], has attracted significant attention over recent years. This task becomes hard already for detecting k-secluded disconnected sets satisfying very simple properties. In particular, detecting a k-secluded independent set of size s is W[1]-hard when parameterized by $k + s$ [1].

Golovach, Heggernes, Lima, and Montealegre [17] suggested then to focus on *connected* k-secluded subgraphs and studied the problem of finding one, which belongs to a graph class $\mathcal{H}$, of maximum total weight. They therefore studied the Connected k-secluded $\mathcal{F}$-free subgraph problem for a finite family $\mathcal{F}$ of forbidden induced subgraphs. Given an undirected graph G in which each vertex v has a positive integer weight $w(v)$, and an integer k, the problem is to find a maximum-weight connected k-secluded vertex set C for which $G[C]$ is $\mathcal{F}$-free. They presented an algorithm based on recursive understanding to solve the problem in time $2^{2^{2^{\mathcal{O}_{\mathcal{F}}(k \log k)}}} \cdot n^{\mathcal{O}_{\mathcal{F}}(1)}$. We improve the dependency on k to single-exponential.

▶ **Corollary 4.** *For each fixed finite family $\mathcal{F}$, Connected k-secluded $\mathcal{F}$-free subgraph can be solved in time $2^{\mathcal{O}_{\mathcal{F}}(k)} \cdot n^{||\mathcal{F}||+\mathcal{O}(1)}$ and polynomial space.*

This result follows directly from Corollary 3 since a maximum-weight k-secluded $\mathcal{F}$-free subgraph must be seclusion-maximal. Hence it suffices to check for each enumerated subgraph whether it is $\mathcal{F}$-free, and remember the heaviest one for which this is the case.

The parameter dependence of our algorithm for Connected k-secluded $\mathcal{F}$-free subgraph is optimal under ETH. This follows from an easy reduction from Maximum Independent Set, which cannot be solved in time $2^{o(n)}$ under ETH [9, Thm. 14.6]. Finding a maximum independent set in an n-vertex graph G is equivalent to finding a maximum-weight triangle-free connected induced ($k = n$)-secluded subgraph in the graph G' that is obtained from G by inserting a universal vertex of weight n and setting the weights of all other vertices to 1. Consequently, an algorithm with running time $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ for Connected k-secluded triangle-free induced subgraph would violate ETH and our parameter dependence is already optimal for $\mathcal{F} = \{K_3\}$.

Application II: Deletion to scattered graph classes

When there are several distinct graph classes (e.g., split graphs and claw-free graphs) on which a problem of interest (e.g. Vertex Cover) becomes tractable, it becomes relevant to compute a minimum vertex set whose removal ensures that each resulting component belongs to one such tractable class. This can lead to fixed-parameter tractable algorithms for solving the original problem on inputs which are *close* to such so-called *islands of tractability* [16]. The corresponding optimization problem has been coined the deletion problem to *scattered* graph classes [21, 23]. Jacob, Majumdar, and Raman [22] (later joined by de Kroon for the journal version [21]) consider the $(\Pi_1, \ldots, \Pi_d)$-deletion problem; given hereditary graph

classes $\Pi_1, \ldots, \Pi_d$, find a set $X \subseteq V(G)$ of at most k vertices such that each connected component of $G - X$ belongs to Π_i for some $i \in [d]$. Here d is seen as a constant. When the set of forbidden induced subgraphs $\mathcal{F}_i$ of Π_i is finite for each $i \in [d]$, they show [21, Lem. 12] that the problem is solvable in time $2^{q(k)+1} \cdot n^{\mathcal{O}_\Pi(1)}$, where $q(k) = 4k^{10(pd)^2+4} + 1$. Here p is the maximum number of vertices of any forbidden induced subgraph.

Using Theorem 2 as a black box, we obtain a single-exponential algorithm for this problem.

▶ **Theorem 5.** *$(\Pi_1, \ldots, \Pi_d)$-DELETION can be solved in time $2^{\mathcal{O}_\Pi(k)} \cdot n^{\mathcal{O}_\Pi(1)}$ and polynomial space when each graph class Π_i is characterized by a finite set $\mathcal{F}_i$ of (not necessarily connected) forbidden induced subgraphs.*

The main idea behind the algorithm is the following. For an arbitrary vertex v, either it belongs to the solution, or we may assume that in the graph that results by removing the solution, the vertex v belongs to a connected component that forms a seclusion-maximal connected k-secluded $\mathcal{F}_i$-free induced subgraph of G for some $i \in [d]$. Branching on each of the $2^{\mathcal{O}_\Pi(k)}$ options gives the desired running time by exploiting the fact that in most recursive calls, the parameter decreases by more than a constant (cf. [9, Thm. 8.19]). Prior to our work, single-exponential algorithms were only known for a handful of ad-hoc cases where $d = 2$, such as deleting to a graph in which each component is a tree or a clique [21], or when one of the sets of forbidden induced subgraphs $\mathcal{F}_i$ contains a path.

Similarly as our first application, the resulting algorithm for $(\Pi_1, \ldots, \Pi_d)$-DELETION is ETH-tight: the problem is a strict generalization of k-VERTEX COVER, which is known not to admit an algorithm with running time $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ unless ETH fails.

Techniques

The proof of Theorem 2 is based on a bounded-depth search tree algorithm with a nontrivial progress measure. By adding vertices to S or T in branching steps of the enumeration algorithm, the sets grow and the size of a minimum (S,T)-separator increases accordingly. The size of a minimum (S,T)-separator disjoint from S is an important progress measure for the algorithm: if it ever exceeds k, there can be no k-secluded set containing all of S and none of T and therefore the enumeration is finished.

The branching steps are informed by the farthest minimum (S,T)-separator (see Lemma 9), similarly as the enumeration algorithm for important separators, but are significantly more involved because we have to handle the forbidden induced subgraphs. A distinctive feature of our algorithm is that the decision made by branching can be to add certain vertices to the set T, while the important-separator enumeration only branches by enriching S. A key step is to use submodularity to infer that a certain vertex set is contained in *all* seclusion-maximal secluded subgraphs under consideration when other branching steps are inapplicable.

As an illustrative example consider the case $\mathcal{F} = \{K_3\}$, that is, we want to enumerate seclusion-maximal vertex sets $C \subseteq V(G) \setminus T$, $C \supseteq S$, which induce connected triangle-free subgraphs with at most k neighbors. Let $\lambda^{\mathrm{L}}(S,T)$ denote the size of a minimum vertex set disjoint from S that separates T from S – we will refer to such separators as *left-restricted*. Then $\lambda^{\mathrm{L}}(S,T)$ corresponds to the minimum possible size of $N(C)$. Similarly to the enumeration algorithm for important separators, we keep track of two measures: (M1) the value of k, and (M2) the gap between k and $\lambda^{\mathrm{L}}(S,T)$. We combine them into a single progress measure which is bounded by $2k$ and decreases during branching.

The first branching scenario occurs when there is some triangle in the graph G which intersects or is adjacent to S; then we guess which of its vertices should belong to $N(C)$, remove it from the graph, and decrease k by one. Otherwise, let $\mathcal{U} = \{U_1, \ldots U_d\}$ be the collection of all vertex sets of triangles in G (which are now disjoint from S). When there exists a triangle U_i whose addition to T increases the value $\lambda^{\mathrm{L}}(S,T)$, we branch into two

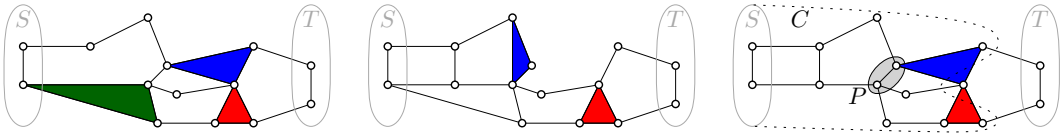


Figure 1 Illustration of the branching steps for enumerating triangle-free k-secluded subgraphs for $k = 3$. Left: the green triangle intersects S; we branch to guess which vertex belongs to $N(C)$. Middle: setting where $2 = \lambda^{\mathrm{L}}(S,T) < \lambda^{\mathrm{L}}(S, T \cup V(\mathcal{U})) = 3$; adding the top triangle to T increases λ^{L}. The set $\mathcal{U}$ consists of the colored triangles. Right: setting where $\lambda^{\mathrm{L}}(S,T) = \lambda^{\mathrm{L}}(S, T \cup V(\mathcal{U})) = 2$, with a corresponding farthest separator P. In this case every seclusion-maximal triangle-free set $C \supseteq S$ must be a superset of the reachability set of S in $G - P$.

possibilities: either U_i is disjoint from $N[C]$ – then we set $T \leftarrow T \cup U_i$ so the measure (M2) decreases – or U_i intersects $N(C)$ – then we perform branching as above. We show that in the remaining case all the triangles are separated from S by the minimum left-restricted (S,T)-separator closest to S; hence the value of $\lambda^{\mathrm{L}}(S,T)$ equals the value of $\lambda^{\mathrm{L}}(S, T \cup V(\mathcal{U}))$. Next, let P be the farthest minimum left-restricted $(S, T \cup V(\mathcal{U}))$-separator; we use submodularity to justify that we can now safely add to S all the vertices reachable from S in $G - P$. This allows us to assume that when $u \in P$ then either $u \in N(C)$ or $u \in C$, which leads to the last branching strategy. We either delete u (so k drops) or add u to S; note that in this case the progress measure may not change directly. The key observation is that adding u to S invalidates the farthest $(S, T \cup V(\mathcal{U}))$-separator P and now we are promised to make progress in the very next branching step. The different branching scenarios are illustrated in Figure 1.

The only property of K_3 that we have relied on is connectivity: if a triangle intersects a triangle-free set C then it must intersect $N(C)$ as well. This is no longer true when $\mathcal{F}$ contains a disconnected graph. For example, the forbidden family for the class of split graphs includes $2K_2$. A subgraph of $F \in \mathcal{F}$ that can be obtained by removing some components from F is called a *partial forbidden graph*. We introduce a third measure to keep track of how many different partial forbidden graphs appear as induced subgraph in $G[S]$. The main difficulty in generalizing the previous approach lies in justification of the greedy argument: when P is a farthest minimum separator between S and a certain set then we want to replace S with the set S' of vertices reachable from S in $G - P$. In the setting of connected obstacles this fact could be proven easily because S' was disjoint from all the obstacles. The problem is now it may contain some partial forbidden subgraphs. We handle this issue by defining P in such a way that the sets of partial forbidden graphs appearing in $G[S]$ and $G[S']$ are the same and giving a rearrangement argument about subgraph isomorphisms. This allows us to extend the analysis to any family $\mathcal{F}$ of forbidden subgraphs.

Organization. We begin with formal preliminaries in Section 2, including proofs of several properties of extremal separators. Next, we present the algorithm for enumerating secluded $\mathcal{F}$-free subgraphs in Section 3 and conclude in Section 4. The proofs of claims indicated with (★) can be found in the full version of the article [24]. The applications of the main theorem are discussed in the full version as well.

2 Preliminaries

Graphs and separators

We consider finite, simple, undirected graphs. We denote the vertex and edge sets of a graph G by $V(G)$ and $E(G)$ respectively, with $|V(G)| = n$ and $|E(G)| = m$. For a set of vertices $S \subseteq V(G)$, by $G[S]$ we denote the graph induced by S. We use shorthand $G - v$

and $G - S$ for $G[V(G) \setminus \{v\}]$ and $G[V(G) \setminus S]$, respectively. The open neighborhood $N_G(v)$ of $v \in V(G)$ is defined as $\{u \in V(G) \mid \{u, v\} \in E(G)\}$. The closed neighborhood of v is $N_G[v] = N_G(v) \cup \{v\}$. For $S \subseteq V(G)$, we have $N_G[S] = \bigcup_{v \in S} N_G[v]$ and $N_G(S) = N_G[S] \setminus S$. The set C is called connected if the graph $G[C]$ is connected.

We proceed by introducing notions concerning separators which are crucial for the branching steps of our algorithms. For two sets $S, T \subseteq V(G)$ in a graph G, a set $P \subseteq V(G)$ is an unrestricted (S,T)-separator if no connected component of $G - P$ contains a vertex from both $S \setminus P$ and $T \setminus P$. Note that such a separator may intersect $S \cup T$. Equivalently, P is an (S,T)-separator if each (S,T)-path contains a vertex of P. A restricted (S,T)-separator is an unrestricted (S,T)-separator P which satisfies $P \cap (S \cup T) = \emptyset$. A left-restricted (S,T)-separator is an unrestricted (S,T)-separator P which satisfies $P \cap S = \emptyset$. Let $\lambda^{\mathrm{L}}_G(S,T)$ denote the minimum size of a left-restricted (S,T)-separator, or $+\infty$ if no such separator exists (which happens when $S \cap T \neq \emptyset$).

▶ **Theorem 6** (Ford-Fulkerson). *There is an algorithm that, given an n-vertex m-edge graph $G = (V, E)$, disjoint sets $S, T \subseteq V(G)$, and an integer k, runs in time $\mathcal{O}(k(n+m))$ and determines whether there exists a restricted (S,T)-separator of size at most k. If so, then the algorithm returns a separator of minimum size.*

By the following observation we can translate properties of restricted separators into properties of left-restricted separators.

▶ **Observation 7.** *Let G be a graph and $S, T \subseteq V(G)$. Consider the graph G' obtained from G by adding a new vertex t adjacent to each $v \in T$. Then $P \subseteq V(G)$ is a left-restricted (S,T)-separator in G if and only if P is a restricted (S,t)-separator in G'.*

Extremal separators and submodularity

The following submodularity property of the cardinality of the open neighborhood is well-known; cf. [40, §44.12] and [28, Fn. 3].

▶ **Lemma 8** (Submodularity). *Let G be a graph and $A, B \subseteq V(G)$. Then the following holds:*

$$|N_G(A)| + |N_G(B)| \geq |N_G(A \cap B)| + |N_G(A \cup B)|.$$

For a graph G and vertex sets $S, P \subseteq V(G)$, we denote by $R_G(S,P)$ the set of vertices which can be reached in $G - P$ from at least one vertex in the set $S \setminus P$.

▶ **Lemma 9.** *Let G be a graph and $S, T \subseteq V(G)$ be two disjoint non-adjacent vertex sets. There exist minimum restricted (S,T)-separators P^- (closest) and P^+ (farthest), such that for each minimum restricted (S,T)-separator P, it holds that $R_G(S,P^-) \subseteq R_G(S,P) \subseteq R_G(S,P^+)$. Moreover, if a minimum restricted (S,T)-separator has size k, then P^- and P^+ can be identified in $\mathcal{O}(k(n+m))$ time.*

Proof. It is well-known (cf. [9, Thm. 8.5] for the edge-based variant of this statement, or [28, §3.2] for the same concept with slightly different terminology) that the existence of these separators follows from submodularity (Lemma 8), while they can be computed by analyzing the residual network when applying the Ford-Fulkerson algorithm to compute a minimum separator. We sketch the main ideas for completeness.

By merging S into a single vertex s^+ and merging T into a single vertex t^-, which is harmless because a restricted separator is disjoint from $S \cup T$, we may assume that S and T are singletons. Transform G into an edge-capacitated directed flow network D in which s^+ is the source and t^- is the sink. All remaining vertices $v \in V(G) \setminus (S \cup T)$ are split into two

representatives v^-, v^+ connected by an arc (v^-, v^+) of capacity 1. For each edge $uv \in E(G)$ with $u, v \in V(G) \setminus \{s^+, t^-\}$ we add arcs $(u^+, v^-), (u^-, v^+)$ of capacity 2. For edges of the form s^+v we add an arc (s^+, v^-) of capacity 2 to D. Similarly, for edges of the form t^-v we add an arc (v^+, t^-) of capacity 2. Then the minimum size k of a restricted (S,T)-separator in G equals the maximum flow value in the constructed network, which can be computed by k rounds of the Ford-Fulkerson algorithm. Each round can be implemented to run in time $\mathcal{O}(n+m)$. From the state of the residual network when Ford-Fulkerson terminates we can extract P^- and P^+ as follows: the set P^- contains all vertices $v \in V(G) \setminus (S \cup T)$ for which the source can reach v^- but not v^+ in the final residual network. Similarly, P^+ contains all vertices $v \in V(G) \setminus (S \cup T)$ for which v^+ can reach the sink but v^- cannot. ◀

By Observation 7, we can apply the lemma above for left-restricted separators too; when the sets S, T are disjoint, then S is non-adjacent to t in the graph obtained by adding a vertex t adjacent to every vertex in T.

The extremal separators identified in Lemma 9 explain when adding a vertex to S or T increases the separator size. The following statement is not symmetric because we work with the non-symmetric notion of a left-restricted separator.

▶ **Lemma 10.** *Let G be a graph, let S, T be disjoint vertex sets, and let P^- and P^+ be the closest and farthest minimum* left-restricted *(S,T)-separators. Then for any vertex $v \in V(G)$, the following holds:*

1. $\lambda^{\mathrm{L}}_G(S \cup \{v\}, T) > \lambda^{\mathrm{L}}_G(S,T)$ *if and only if* $v \in R_G(T, P^+) \cup P^+$.
2. $\lambda^{\mathrm{L}}_G(S, T \cup \{v\}) > \lambda^{\mathrm{L}}_G(S,T)$ *if and only if* $v \in R_G(S, P^-)$.

Proof. Adding a vertex to S or T can never decrease the separator size, so for both cases, the left-hand side is either equal to or strictly greater than the right-hand side.

(1). Observe that if $v \notin R_G(T, P^+) \cup P^+$, then P^+ is also a left-restricted $(S \cup \{v\}, T)$-separator which implies $\lambda^{\mathrm{L}}_G(S \cup \{v\}, T) = \lambda^{\mathrm{L}}_G(S,T)$. If $v \in T$, then (1) holds as $\lambda^{\mathrm{L}}_G(S \cup \{v\}, T) = +\infty$. Consider now $v \in (R_G(T, P^+) \cup P^+) \setminus T$; we argue that adding it to S increases the separator size. Assume for a contradiction that there exists a minimum left-restricted $(S \cup \{v\}, T)$-separator P of size at most $\lambda^{\mathrm{L}}_G(S,T) = |P^+|$. Note that since P is left-restricted, we have $v \notin P$. Observe that P is also a left-restricted (S,T)-separator. By Lemma 9 we have $R_G(S,P) \subseteq R_G(S, P^+)$. Since $v \in (R_G(T, P^+) \cup P^+) \setminus T$, it follows that $v \notin R_G(S,P)$. We do a case distinction on v to construct a path Q from v to T.

- In the case that $v \in P^+ \setminus T$, then since P^+ is a minimum separator it must be inclusion-minimal. Therefore, since $P^+ \setminus \{v\}$ is not an (S,T)-separator, it follows that v has a neighbor in $R_G(T, P^+)$ and so there is a path Q from v to T in the graph induced by $R_G(T, P^+) \cup \{v\}$ such that $V(Q) \cap P^+ = \{v\}$.
- In the case that $v \in R_G(T, P^+) \setminus T$, then by definition there is a path from v to T in the graph induced by $R_G(T, P^+)$.

Since P is a left-restricted $(S \cup \{v\}, T)$-separator and therefore $v \notin P$, it follows that P contains at least one vertex $u \in V(Q)$ that is not in $R_G(S, P^+) \cup P^+$. Let P' be the set of vertices adjacent to $R_G(S,P)$. Since all vertices of P' belong to P while $u \notin P'$, it follows that P' is a left-restricted (S,T)-separator that is strictly smaller than P, a contradiction to $|P| \leq \lambda^{\mathrm{L}}_G(S,T)$.

(2). If $v \notin R_G(S, P^-)$, then P^- is a left-restricted $(S, T \cup \{v\})$-separator as well which implies $\lambda^{\mathrm{L}}_G(S, T \cup \{v\}) = \lambda^{\mathrm{L}}_G(S, T)$. If $v \in R_G(S, P^-)$, suppose that there exists a minimum left-restricted $(S, T \cup \{v\})$-separator P of size $|P^-|$. Note that $v \notin S$, as otherwise no such separator exists. Furthermore P is also a left-restricted (S, T)-separator. By Lemma 9 we have $R_G(S, P^-) \subseteq R_G(S, P)$. But since $v \notin R_G(S, P)$ we reach a contradiction as $R_G(S, P) \not\supseteq R_G(S, P^-)$. ◀

The following lemma captures the idea that if $\lambda^{\mathrm{L}}_G(S, T \cup Z) > \lambda^{\mathrm{L}}_G(S, T)$, then there is a single vertex from Z whose addition to T already increases the size of a minimum left-restricted (S, T)-separator. We will use it to argue that when it is cheaper to separate S from T than to separate S from T together with all obstacles of a certain form, then there is already a single vertex from one such obstacle which causes this increase.

▶ **Lemma 11.** *Let G be a graph, $S \subseteq V(G)$, and $T, Z \subseteq V(G) \setminus S$. If there is no vertex $v \in Z$ such that $\lambda^{\mathrm{L}}_G(S, T \cup \{v\}) > \lambda^{\mathrm{L}}_G(S, T)$, then $\lambda^{\mathrm{L}}_G(S, T) = \lambda^{\mathrm{L}}_G(S, T \cup Z)$. Furthermore if $\lambda^{\mathrm{L}}_G(S, T) \leq k$, then in $\mathcal{O}(k(n + m))$ time we can either find such a vertex v or determine that no such vertex exists.*

Proof. Let P^- be the minimum left-restricted (S, T)-separator which is closest to S. If for every $v \in Z$ the value of $\lambda^{\mathrm{L}}_G(S, T \cup \{v\})$ equals $\lambda^{\mathrm{L}}_G(S, T)$ then Lemma 10 implies that each $v \in Z$ lies outside $R_G(S, P^-)$ so $Z \cap R_G(S, P^-) = \emptyset$. Then P^- is a left-restricted $(S, T \cup Z)$-separator of size $\lambda^{\mathrm{L}}_G(S, T)$.

On the other hand, if there is a vertex $v \in Z$ for which $\lambda^{\mathrm{L}}_G(S, T \cup \{v\}) > \lambda^{\mathrm{L}}_G(S, T)$ then $v \in R_G(S, P^-)$. Hence, in order to detect such a vertex it suffices to compute the closest minimum left-restricted (S, T)-separator P^-, which can be done in time $\mathcal{O}(k(n + m))$ via Lemma 9. ◀

Finally, the last lemma of this section uses submodularity to argue that the neighborhood size of a vertex set C with $S \subseteq C \subseteq V(G) \setminus T$ does not increase when taking its union with the reachable set $R_G(S, P)$ with respect to a minimum left-restricted (S, T)-separator P.

▶ **Lemma 12.** *If $P \subseteq V(G)$ is a minimum left-restricted (S, T)-separator in a graph G and $S' = R_G(S, P)$, then for any set C with $S \subseteq C \subseteq V(G) \setminus T$ we have $|N_G(C \cup S')| \leq |N_G(C)|$.*

Proof. Observe that since P is a minimum left-restricted (S, T)-separator, we have $|P| = \lambda^{\mathrm{L}}_G(S, T)$ and $P = N_G(S')$. We apply the submodular inequality to the sets C and S'.

$$|N_G(C)| + |N_G(S')| \geq |N_G(C \cup S')| + |N_G(C \cap S')| \geq |N_G(C \cup S')| + \lambda^{\mathrm{L}}_G(S, T).$$

Here the last step comes from the fact that $S \subseteq S' \subseteq V(G) \setminus T$ since it is the set reachable from S with respect to a left-restricted (S, T)-separator, so that $C \cap S'$ contains all of S and is disjoint from T. This implies that $N_G(C \cap S')$ is a left-restricted (S, T)-separator, so that $|N_G(C \cap S')| \geq \lambda^{\mathrm{L}}_G(S, T)$.

As $|N_G(S')| = |P| = \lambda^{\mathrm{L}}_G(S, T)$, canceling these terms from both sides gives $|N_G(C)| \geq |N_G(C \cup S')|$ which completes the proof. ◀

3 The enumeration algorithm

We need the following concept to deal with forbidden subgraphs which may be disconnected.

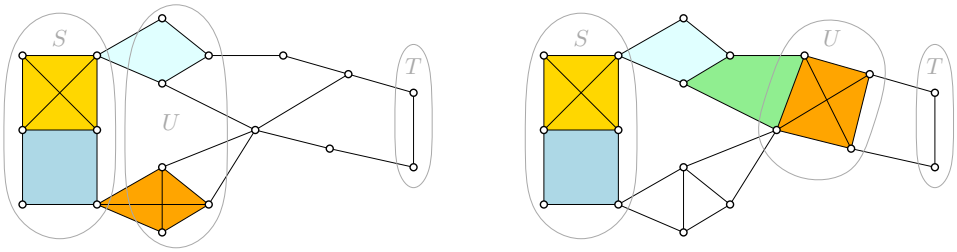


Figure 2 Illustration of the idea of enrichment and the branching steps in the proof of Theorem 2. Here $F = C_4 \uplus K_4$. Left: The graph $G[S]$ contains C_4 and K_4, but not F. The set U enriches S since $G[S \cup U]$ contains a new partial forbidden graph F. Every component of $G[U]$ is adjacent to S, so Step 3 applies. Right: The two top copies of C_4 do not enrich S. One of them intersects the only copy of K_4 in $G[S]$; the other one is adjacent to the only copy of K_4, while F has to appear as an induced subgraph. However the connected set U enriches S and it gets detected in Step 4. In both cases the enrichments are tight.

▶ **Definition 13.** *A partial forbidden graph F' is a graph obtained from some $F \in \mathcal{F}$ by deleting zero or more connected components. (So each $F \in \mathcal{F}$ itself is also considered a partial forbidden graph.)*

We use the following notation to work with induced subgraph isomorphisms. An induced subgraph isomorphism from H to G is an injection $\phi\colon V(H) \to V(G)$ such that for all distinct $u, v \in V(H)$ we have $\{u, v\} \in E(H)$ if and only if $\{\phi(u), \phi(v)\} \in E(G)$. For a vertex set $U \subseteq V(H)$ we let $\phi(U) := \{\phi(u) \mid u \in U\}$. For a subgraph H' of H we write $\phi(H')$ instead of $\phi(V(H'))$.

The following definition will be important to capture the progress of the recursive algorithm. See Figure 2 for an illustration.

▶ **Definition 14.** *We say that a vertex set $U \subseteq V(G)$ enriches a vertex set $S \subseteq V(G)$ with respect to $\mathcal{F}$ if there exists a partial forbidden graph F' such that $G[S \cup U]$ contains an induced subgraph isomorphic to F' but $G[S]$ does not. We call such a set U an enrichment.*

An enrichment U is called tight if $U = \phi(F') \setminus S$ for some induced subgraph isomorphism $\phi\colon V(F') \to V(G)$ from some partial forbidden graph F' for which $G[S]$ does not contain an induced subgraph isomorphic to F'.

The following observation will be used to argue for the correctness of the recursive scheme. Note that we get an implication only in one way (being seclusion-maximal in G implies being seclusion-maximal in $G - v$, not the other way around), which is the reason why we output a superset of the sought set in Theorem 2.

▶ **Observation 15.** *Let G be a graph containing disjoint sets $S, T \subseteq V(G)$ and let $C \subseteq V(G)$ be seclusion-maximal with respect to being connected, $\mathcal{F}$-free, k-secluded and satisfying $S \subseteq C \subseteq V(G) \setminus T$. For each $v \in N_G(C)$ it holds that C is seclusion-maximal in $G - v$ with respect to being connected, $\mathcal{F}$-free, $(k-1)$-secluded and satisfying $S \subseteq C \subseteq V(G - v) \setminus T$.*

With these ingredients, we present the enumeration algorithm. Recall that $||\mathcal{F}|| = \max_{F \in \mathcal{F}} |V(F)|$ denotes the maximum order of any graph in $\mathcal{F}$.

▶ **Theorem 2.** *Let $\mathcal{F}$ be a finite set of graphs. For any n-vertex graph G, non-empty vertex set $S \subseteq V(G)$, potentially empty $T \subseteq V(G) \setminus S$, and integer k, the number of k-secluded induced subgraphs $G[C]$ which are seclusion-maximal with respect to being connected, $\mathcal{F}$-free, and satisfying $S \subseteq C \subseteq V(G) \setminus T$, is bounded by $2^{\mathcal{O}_{\mathcal{F}}(k)}$. A superset of size $2^{\mathcal{O}_{\mathcal{F}}(k)}$ of these subgraphs can be enumerated in time $2^{\mathcal{O}_{\mathcal{F}}(k)} \cdot n^{||\mathcal{F}||+\mathcal{O}(1)}$ and polynomial space.*

Proof. Algorithm $\mathsf{Enum}_{\mathcal{F}}(G, S, T, k)$ solves the enumeration task as follows.

1. Stop the algorithm if one of the following holds:
   - **a.** $\lambda^{\mathrm{L}}_G(S, T) > k$,
   - **b.** the vertices of S are not contained in a single connected component of G, or
   - **c.** the graph $G[S]$ contains an induced subgraph isomorphic to some $F \in \mathcal{F}$.

   There are no secluded subgraphs satisfying all imposed conditions.
2. If the connected component C of G which contains S is $\mathcal{F}$-free and includes no vertex of T: output C and stop.

   Component C is the unique seclusion-maximal one satisfying the imposed conditions.
3. If there is a vertex set $U \subseteq V(G) \setminus (S \cup T)$ such that:
   - each connected component of $G[U]$ is adjacent to a vertex of S, and
   - the set U is a tight enrichment of S with respect to $\mathcal{F}$ (so $G[S \cup U]$ contains a new partial forbidden graph)

   then execute the following calls and stop:
   - **a.** For each $u \in U$ call $\mathsf{Enum}_{\mathcal{F}}(G - u, S, T, k - 1)$.
   - **b.** Call $\mathsf{Enum}_{\mathcal{F}}(G, S \cup U, T, k)$.

   A tight enrichment can have at most $||\mathcal{F}||$ vertices which bounds the branching factor in Step 3a. Note that these are exhaustive even though we do not consider adding U to T: since each component of $G[U]$ is adjacent to a vertex of S, if a relevant secluded subgraph does not contain all of U then it contains some vertex of U in its neighborhood and we find it in Step 3a.
4. For the rest of the algorithm, let $\mathcal{U}$ denote the collection of all connected vertex sets $U \subseteq V(G) \setminus (S \cup T)$ which form tight enrichments of S with respect to $\mathcal{F}$. Let $V(\mathcal{U}) := \bigcup_{U \in \mathcal{U}} U$.
   - **a.** If $\lambda^{\mathrm{L}}_G(S, T) < \lambda^{\mathrm{L}}_G(S, T \cup V(\mathcal{U}))$: then (using Lemma 11) there exists $U \in \mathcal{U}$ such that $\lambda^{\mathrm{L}}_G(S, T \cup U) > \lambda^{\mathrm{L}}_G(S, T)$, execute the following calls and stop:
     - **i.** For each $u \in U$ call $\mathsf{Enum}_{\mathcal{F}}(G - u, S, T, k - 1)$. (The value of k decreases.)
     - **ii.** Call $\mathsf{Enum}_{\mathcal{F}}(G, S \cup U, T, k)$. (We absorb a new partial forbidden graph.)
     - **iii.** Call $\mathsf{Enum}_{\mathcal{F}}(G, S, T \cup U, k)$. (The separator size increases.)
   - **b.** If $\lambda^{\mathrm{L}}_G(S, T) = \lambda^{\mathrm{L}}_G(S, T \cup V(\mathcal{U}))$, then let P be the farthest left-restricted minimum $(S, T \cup V(\mathcal{U}))$-separator in G, and let $S' = R_G(S, P) \supseteq S$. Pick an arbitrary $p \in P$ (which may be contained in T but not in S).
     - **i.** Call $\mathsf{Enum}_{\mathcal{F}}(G - p, S', T \setminus \{p\}, k - 1)$. (The value of k decreases.)
     - **ii.** If $p \notin T$, then call $\mathsf{Enum}_{\mathcal{F}}(G, S' \cup \{p\}, T, k)$.
       (Either here or in the next iteration we will be able to make progress.)

   It might happen that $\mathcal{U}$ is empty; in this case the algorithm will execute Step 4b. Also note that P is non-empty because the algorithm did not stop in Step 2; hence it is always possible to choose a vertex $p \in P$.

Before providing an in-depth analysis of the algorithm, we establish that it always terminates. For each recursive call, either a vertex outside S is deleted, or one of S or T grows in size while the two remain disjoint. Since S and T are vertex subsets of a finite graph, this process terminates. The key argument in the correctness of the algorithm is formalized in the following claim.

▷ Claim 16. If the algorithm reaches Step 4b, then every seclusion-maximal k-secluded subgraph satisfying the conditions of the theorem statement contains S'.

Proof. We prove the claim by showing that for an arbitrary k-secluded $\mathcal{F}$-free connected induced subgraph $G[C]$ satisfying $S \subseteq C \subseteq V(G) \setminus T$, the subgraph induced by $C \cup S'$ also satisfies these properties while $|N_G(C \cup S')| \leq |N_G(C)|$. Hence any seclusion-maximal subgraph satisfying the conditions contains S'.

Under the conditions of Step 4b, we have $\lambda^{\mathrm{L}}_G(S,T) = \lambda^{\mathrm{L}}_G(S, T \cup V(\mathcal{U}))$, so that the set P is a left-restricted minimum (S,T)-separator. Next, we have $S' = R_G(S,P)$. By exploiting submodularity of the size of the open neighborhood, we prove in Lemma 12 that $|N_G(C \cup S')| \leq |N_G(C)|$. The key part of the argument is to prove that $C \cup S'$ induces an $\mathcal{F}$-free subgraph. Assume for a contradiction that $G[C \cup S']$ contains an induced subgraph isomorphic to $F \in \mathcal{F}$ and let $\phi\colon V(F) \to C \cup S'$ denote an induced subgraph isomorphism. Out of all ways to choose ϕ, fix a choice that minimizes the number of vertices $|\phi(F) \setminus S|$ the subgraph uses from outside S. We distinguish two cases.

Neighborhood of S intersects $\phi(F)$

If $\phi(F) \cap N_G(S) \neq \emptyset$, then we will use the assumption that Step 3 of the algorithm was not applicable to derive a contradiction. Let F' be the graph consisting of those connected components F_i of F for which $\phi(F_i) \cap N_G[S] \neq \emptyset$; let $U = \phi(F') \setminus S$. Observe that each connected component of $G[U]$ is adjacent to a vertex of S. By construction U is disjoint from S, and U is disjoint from T since $\phi(F) \subseteq C \cup S'$ while both these sets are disjoint from T. Hence U satisfies all but one of the conditions for applying Step 3. Since the algorithm reached Step 4b, it follows that U failed the last criterion which means that the partial forbidden graph F' also exists as an induced subgraph in $G[S]$. Let $\phi_{F'}\colon V(F') \to S$ be an induced subgraph isomorphism from F' to $G[S]$. Since all vertices $v \in V(F)$ for which $\phi(v) \in N_G[S]$ satisfy $v \in V(F')$, we can define a new subgraph isomorphism ϕ' of F in $G[C \cup S']$ as follows for each $v \in V(F)$:

$$\phi'(v) = \begin{cases} \phi_{F'}(v) & \text{if } v \in F' \\ \phi(v) & \text{otherwise.} \end{cases} \tag{1}$$

Observe that this is a valid induced subgraph isomorphism since F' consists of some connected components of F, and we effectively replace the model of F' by $\phi_{F'}$. Since the model of the remaining graph $\overline{F'} = F - F'$ does not use any vertex of $N_G[S]$ by definition of F', there are no edges between vertices of $\phi_{F'}(F')$ and vertices of $\phi(\overline{F'})$, which validates the induced subgraph isomorphism.

Since $\phi(F)$ contains at least one vertex from $N_G(S)$ while $\phi'(F)$ does not, and the only vertices of $\phi'(F) \setminus \phi(F)$ belong to S, we conclude that $\phi'(F)$ contains strictly fewer vertices outside S than $\phi(F)$; a contradiction to minimality of ϕ.

Neighborhood of S does not intersect $\phi(F)$

Now suppose that $\phi(F) \cap N_G(S) = \emptyset$. If $\phi(F) \subseteq C$, then $\phi(F)$ is an induced F-subgraph in $G[C]$, a contradiction to the assumption that C is $\mathcal{F}$-free. Hence $\phi(F)$ must contain a vertex $v \in S' \setminus C \subseteq S' \setminus S$. Since the previous case was not applicable, $v \notin N_G(S)$ and therefore $v \in S' \setminus N_G[S]$.

Fix an arbitrary connected component F_i of F for which $\phi(F_i)$ contains a vertex of $S' \setminus N_G[S]$. We derive several properties of $\phi(F_i)$.

1. Since F_i is a connected component of F, the graph $G[\phi(F_i)]$ is connected.
2. We claim that $\phi(F_i) \cap S = \emptyset$. Note that a connected subgraph cannot both contain a vertex from S and a vertex outside $N_G[S]$ without intersecting $N_G(S)$. Since $\phi(F) \cap N_G(S) = \emptyset$ by the case distinction, the graph $G[\phi(F_i)]$ is connected since F_i is connected, and $\phi(F_i)$ contains a vertex of $S' \setminus N_G[S]$, we find $\phi(F_i) \cap S = \emptyset$.
3. $\phi(F_i) \cap T = \emptyset$, since $\phi(F) \subseteq C \cup S'$ while both C and S' are disjoint from T.

4. We claim that $\phi(F_i) \notin \mathcal{U}$. To see that, recall that $S' = R_G(S, P)$ is the set of vertices reachable from S when removing the $(S, T \cup V(\mathcal{U}))$-separator P. The definition of separator therefore ensures that no vertex of S' belongs to $V(\mathcal{U})$. Since $\phi(F_i)$ contains a vertex of $S' \setminus N_G[S]$ by construction, some vertex of $\phi(F_i)$ does not belong to $V(\mathcal{U})$ and therefore $\phi(F_i) \notin \mathcal{U}$.

Now note that $\phi(F_i)$ satisfies almost all requirements for being contained in the set $\mathcal{U}$ defined in Step 4: it induces a connected subgraph and it is disjoint from $S \cup T$. From the fact that $\phi(F_i) \notin \mathcal{U}$ we therefore conclude that it fails the last criterion: the set $\phi(F_i)$ is not a tight enrichment of S.

Let F' be the graph formed by F_i together with all components F_j of F for which $\phi(F_j) \subseteq S$; then $\phi(F_i) = \phi(F') \setminus S$. Since $\phi(F_i)$ is not a tight enrichment of S, the partial forbidden graph F' is also contained in $G[S]$. Let $\phi_{F'} : F' \to S$ denote an induced subgraph isomorphism of F' to $G[S]$. Since $\phi(F)$ contains no vertex of $N_G(S)$, we can define a new subgraph isomorphism ϕ' of F in $G[C \cup S']$ exactly as in (1).

Since the graph F' consists of some connected components of F, while $\phi_{F'}(F') \subseteq S$ and $\phi(\overline{F'}) \cap N_G[S] = \emptyset$, it follows that ϕ' is an induced subgraph isomorphism of F in $G[C \cup S']$. But $|\phi'(F) \setminus S|$ is strictly smaller than $|\phi(F) \setminus S|$ since $\phi(F_i)$ intersects $S' \setminus N_G[S]$ while $\phi'(F_i) \subseteq \phi'(F') \subseteq S$ and ϕ and ϕ' coincide on $\overline{F'}$. This contradicts the minimality of the choice of ϕ.

Since the case distinction is exhaustive, this proves the claim. ◁

Using the previous claim, we can establish the correctness of the algorithm.

▷ Claim 17. If $G[C]$ is an induced subgraph of G that is seclusion-maximal with respect to being connected, $\mathcal{F}$-free, k-secluded and satisfying $S \subseteq C \subseteq V(G) \setminus T$, then C occurs in the output of $\mathsf{Enum}_{\mathcal{F}}(G, S, T, k)$.

Proof. We prove this claim by induction on the recursion depth of the $\mathsf{Enum}_{\mathcal{F}}$ algorithm, which is valid as we argued above it is finite. In the base case, the algorithm does not recurse. In other words, the algorithm either stopped in Step 1 or 2. If the algorithm stops in Step 1, then there can be no induced subgraph satisfying the conditions and so there is nothing to show. If the algorithm stops in Step 2, then the only seclusion-maximal induced subgraph is the $\mathcal{F}$-free connected component containing S. Note that this component is k-secluded since $k \geq 0$ as $\lambda^{\mathrm{L}}_G(S, T) \geq 0$ and the algorithm did not stop in Step 1a.

For the induction step, we may assume that each recursive call made by the algorithm correctly enumerates a superset of the seclusion-maximal subgraphs satisfying the conditions imposed by the parameters of the recursive call, as the recursion depth of the execution of those calls is strictly smaller than the recursion depth for the current arguments (G, S, T, k). Consider a connected $\mathcal{F}$-free k-secluded induced subgraph $G[C]$ of G with $S \subseteq C \subseteq V(G) \setminus T$ that is seclusion-maximal with respect to satisfying all these conditions. Suppose there is a vertex set $U \subseteq V(G) \setminus (S \cup T)$ that satisfies the conditions of Step 3. If $U \subseteq C$, then by induction C is part of the enumerated output of Step 3b. Otherwise, since each connected component of $G[U]$ is adjacent to a vertex in S, there is at least one vertex $u \in U$ such that $u \in N_G(C)$. By Observation 15, the output of the corresponding call in Step 3a contains C. Note that since $U \cap T = \emptyset$, we have $T \subseteq V(G) \setminus (S \cup U)$ and therefore the recursive calls satisfy the input requirements.

Next we consider the correctness in case such a set U does not exist so the algorithm reaches Step 4. Let $\mathcal{U}$ be the set of tight enrichments as defined in Step 4. First suppose that $\lambda^{\mathrm{L}}_G(S, T) < \lambda^{\mathrm{L}}_G(S, T \cup V(\mathcal{U}))$. Then by the contrapositive of the first part of Lemma 11 with $Z = V(\mathcal{U})$, there is a vertex $v \in V(\mathcal{U}) \setminus T$ such that $\lambda^{\mathrm{L}}_G(S, T \cup \{v\}) > \lambda^{\mathrm{L}}_G(S, T)$. By

picking an enrichment $U \in \mathcal{U}$ such that $v \in U$, this implies $\lambda^{\mathrm{L}}_G(S, T \cup U) > \lambda^{\mathrm{L}}_G(S, T)$. Now if there is a vertex $u \in U$ such that $u \in N_G(C)$, then by induction and Observation 15 we get that C is output by the corresponding call in Step 4(a)i. Otherwise, either $U \subseteq C$ or $U \cap C = \emptyset$ (since U is connected) and C is found in Step 4(a)ii or Step 4(a)iii respectively. Again observe that these recursive calls satisfy the input requirements as $U \cap (S \cup T) = \emptyset$.

Finally suppose that $\lambda^{\mathrm{L}}_G(S, T) = \lambda^{\mathrm{L}}_G(S, T \cup V(\mathcal{U}))$. By Claim 16 we get that $S' \subseteq C$. We first argue that $P = N_G(S')$ is non-empty. Note that since the algorithm did not stop in Step 1, the graph $G[S]$ is $\mathcal{F}$-free and S is contained in a single connected component of G. Furthermore since it did not stop in Step 2, the connected component containing S either has a vertex of T or is not $\mathcal{F}$-free. Note that the former case already implies $\lambda^{\mathrm{L}}_G(S, T) > 0$. If the component has no vertex of T and is not $\mathcal{F}$-free, then it contains a vertex set J for which $G[J]$ is isomorphic to some $F \in \mathcal{F}$. Observe that $J \setminus (S \cup T) = J \setminus S$ is a tight enrichment of S. We have established that it is possible to enrich S but we need an enrichment that meets the conditions of Step 4. Let $U \subseteq V(G) \setminus (S \cup T)$ be a tight enrichment of minimum size and let $\phi\colon V(F') \to V(G)$ be the corresponding subgraph isomorphism from some partial forbidden graph F'; we have $U = \phi(F') \setminus S$. We argue that $G[U]$ is connected. If each connected component of $G[U]$ is adjacent to a vertex of S, then Step 3 would have applied, contradicting the fact that the algorithm reaches Step 4. Hence, there exists a connected component of $G[U]$ that is non-adjacent to S; let U' be the vertex set of such a component. Since U is chosen to be minimum, we get that $U \setminus U'$ is not a tight enrichment, and so there is an induced subgraph of $G[S]$ isomorphic to the partial forbidden graph $F'' = G[\phi(F') \setminus U']$. This subgraph of $G[S]$ combines with the graph $G[U']$ to form an induced subgraph isomorphic to F' (we exploit that U' is not adjacent to S), which shows that U' is a tight enrichment. By minimality of U we obtain $U = U'$. Hence U is not adjacent to S and the graph $G[U]$ is connected so $U \in \mathcal{U}$. Since U and S are contained in the same connected component we get that $\lambda^{\mathrm{L}}_G(S, T \cup V(\mathcal{U})) > 0$. This implies there exists some vertex $p \in P = N_G(S')$. Since $S' \subseteq C$, we either get $p \in N_G(C)$, or (if $p \notin T$) $p \in C$. By induction (and Observation 15) we conclude that C is part of the output of Step 4(b)i or Step 4(b)ii. The condition $p \notin T$ ensures that the input requirements of the latter recursive call are satisfied. ◁

As the previous claim shows that the algorithm enumerates a superset of the relevant seclusion-maximal induced subgraphs, to prove Theorem 2 it suffices to bound the size of the search tree generated by the algorithm, and thereby the running time and total number of induced subgraphs which are given as output. To that end, we argue that for any two successive recursive calls in the recursion tree, at least one of them makes strict progress on a relevant measure. Since no call can increase the measure, this will imply a bound on the depth of the recursion tree. Since it is easy to see that the branching factor is a constant depending on $||\mathcal{F}||$, this will lead to the desired bound.

▷ Claim 18 (★). The search tree generated by the call $\mathsf{Enum}_{\mathcal{F}}(G, S, T, k)$ has depth $\mathcal{O}_{\mathcal{F}}(k)$ and $2^{\mathcal{O}_{\mathcal{F}}(k)}$ leaves.

The previous claim implies that the number of seclusion-maximal connected $\mathcal{F}$-free k-secluded induced subgraphs containing all of S and none of T is $2^{\mathcal{O}_{\mathcal{F}}(k)}$, since the algorithm outputs at most one subgraph per call and only does so in leaf nodes of the recursion tree. As Claim 18 bounds the size of the search tree generated by the algorithm, the desired bound on the total running time follows from the claim below.

▷ Claim 19 (★). A single iteration of $\mathsf{Enum}_{\mathcal{F}}(G, S, T, k)$ can be implemented to run in time $|\mathcal{F}| \cdot 2^{||\mathcal{F}||} \cdot n^{||\mathcal{F}|| + \mathcal{O}(1)}$ and polynomial space.

This concludes the proof of Theorem 2. ◀

4 Conclusion

We have introduced a new algorithmic primitive based on secluded connected subgraphs which generalizes important separators. The high-level idea behind the algorithm is *enumeration via separation*: by introducing an artificial set T and considering the more general problem of enumerating secluded subgraphs containing S but disjoint from T, we can analyze the progress of the recursion in terms of the size of a minimum (left-restricted) (S,T)-separator. We expect this idea to be useful in scenarios beyond the one studied here.

We presented a single-exponential, polynomial-space FPT algorithm to enumerate the family of seclusion-maximal connected $\mathcal{F}$-free subgraphs for finite $\mathcal{F}$, making it potentially viable for practical use [38]. The combination of single-exponential running time and polynomial space usage sets our approach apart from others such as recursive understanding [8, 10, 31] and treewidth reduction [36]. Algorithms exploiting half-integrality of the linear-programming relaxation or other discrete relaxations also have these desirable properties, though [12, 18, 19, 20, 41]. Using this approach, Iwata, Yamaguchi, and Yoshida [20] even obtained a *linear-time* algorithm in terms of the number of vertices n, solving (vertex) Multiway Cut in time $2^k \cdot k \cdot (n+m)$. At a high level, there is some resemblance between their approach and ours. They work on a discrete relaxation of deletion problems in graphs which are not standard LP-relaxations, but are based on relaxations of a *rooted* problem in which only constraints involving a prescribed set S are active. This is reminiscent of the fact that we enumerate secluded subgraphs containing a prescribed set S. Their branching algorithms are based on the notion of an extremal optimal solution to the LP relaxation, which resembles our use of the farthest minimum left-restricted (S,T)-separator. However, the two approaches diverge there. To handle problems via their approach, they should be expressible as a 0/1/ALL CSP. Problems for which the validity of a solution can be verified by unit propagation (such as Node Unique Label Cover, Node Multiway Cut, Subset and Group Feedback Vertex Set) belong to this category, but it seems impossible to express the property of being $\mathcal{F}$-free for arbitrary finite sets $\mathcal{F}$ in this framework.

The branching steps underlying our algorithm were informed by the structure of the subgraphs induced by certain vertex sets. In the considered setting, where certain possibly disconnected structures are not allowed to appear inside C, it is necessary to characterize the forbidden sets in terms of the graph structure they induce. But when the forbidden sets are connected, we believe our proof technique can be used in a more general setting to establish the following. For any n-vertex graph G, non-empty vertex set $S \subseteq V(G)$, potentially empty $T \subseteq V(G) \setminus S$, integer k, and collection $F_1, \ldots, F_m \subseteq V(G)$ of vertex sets of size at most ℓ which are connected in G, the number of k-secluded induced subgraphs $G[C]$ which are seclusion-maximal with respect to being connected, not containing any set F_i, and satisfying $S \subseteq C \subseteq V(G) \setminus T$, is bounded by $(2+\ell)^{\mathcal{O}(k)}$, and a superset of them can be enumerated in time $(2+\ell)^{\mathcal{O}(k)} \cdot m \cdot n^{\mathcal{O}(1)}$ and polynomial space. The reason why dealing with general connected obstacles is feasible is that whenever $F_i \cap C \neq \emptyset$ then also $F_i \cap N(C) \neq \emptyset$; this allows us to always make progress using the simpler branching strategy without keeping track of partial forbidden graphs. The corresponding generalization for *disconnected* vertex sets F_i is false, even for $|F_i| = 2$. To see this, consider a graph consisting of a cycle on $2m+1$ vertices consecutively labeled $s, a_1, \ldots, a_m, b_1, \ldots, b_m$ with $F_i = \{a_i, b_i\}$ for each $i \in [m]$, in which the number of relevant seclusion-maximal 2-secluded sets containing s is $\Omega(m)$.

We leave it to future work to consider generalizations of our ideas to *directed graphs*. Since important separators also apply in that setting, we expect the branching step in terms of left-restricted minimum separators to be applicable in directed graphs as well. However,

there are multiple ways to generalize the notion of a connected secluded induced subgraph to the directed setting: one can consider weak connectivity, strong connectivity, or a rooted variant where we consider all vertices reachable from a source vertex x. Similarly, one can define seclusion in terms of the number of in-neighbors, out-neighbors, or both.

References

1 René van Bevern, Till Fluschnik, George B. Mertzios, Hendrik Molter, Manuel Sorge, and Ondrej Suchý. The parameterized complexity of finding secluded solutions to some classical optimization problems on graphs. *Discret. Optim.*, 30:20–50, 2018. doi:10.1016/j.disopt.2018.05.002.

2 René van Bevern, Till Fluschnik, and Oxana Yu. Tsidulko. Parameterized algorithms and data reduction for the short secluded s-t-path problem. *Networks*, 75(1):34–63, 2020. doi:10.1002/net.21904.

3 Hans L. Bodlaender, Anuj Dawar, and Virginia V. Williams. EATCS-IPEC Nerode Prize 2020, 2020. URL: https://eatcs.org/index.php/component/content/article/1-news/2861-eatcs-ipec-nerode-prize-2020-.

4 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. *SIAM J. Comput.*, 47(1):166–207, 2018. doi:10.1137/140961808.

5 Shiri Chechik, Matthew P. Johnson, Merav Parter, and David Peleg. Secluded connectivity problems. *Algorithmica*, 79(3):708–741, 2017. doi:10.1007/s00453-016-0222-z.

6 Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009. doi:10.1007/s00453-007-9130-6.

7 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008. doi:10.1145/1411509.1411511.

8 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.

9 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.

10 Marek Cygan, Pawel Komosa, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Saket Saurabh, and Magnus Wahlström. Randomized contractions meet lean decompositions. *ACM Trans. Algorithms*, 17(1):6:1–6:30, 2021. doi:10.1145/3426738.

11 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed-parameter tractable. *SIAM J. Comput.*, 48(2):417–450, 2019. doi:10.1137/140988553.

12 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *ACM Trans. Comput. Theory*, 5(1):3:1–3:11, 2013. doi:10.1145/2462896.2462899.

13 Huib Donkers, Bart M. P. Jansen, and Jari J. H. de Kroon. Finding k-secluded trees faster. In *Proceeding of the 48th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2022*, volume 13453 of *Lecture Notes in Computer Science*, pages 173–186. Springer, 2022. doi:10.1007/978-3-031-15914-5_13.

14 Fedor V. Fomin, Petr A. Golovach, Nikolay Karpov, and Alexander S. Kulikov. Parameterized complexity of secluded connectivity problems. *Theory Comput. Syst.*, 61(3):795–819, 2017. doi:10.1007/s00224-016-9717-x.

15 L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.

16 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. *ACM Trans. Algorithms*, 13(2):29:1–29:32, 2017. doi:10.1145/3014587.

17 Petr A. Golovach, Pinar Heggernes, Paloma T. Lima, and Pedro Montealegre. Finding connected secluded subgraphs. *J. Comput. Syst. Sci.*, 113:101–124, 2020. doi:10.1016/j.jcss.2020.05.006.

18 Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discret. Optim.*, 8(1):61–71, 2011. doi:10.1016/j.disopt.2010.05.003.

19 Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, LP-branching, and FPT algorithms. *SIAM J. Comput.*, 45(4):1377–1411, 2016. doi:10.1137/140962838.

20 Yoichi Iwata, Yutaro Yamaguchi, and Yuichi Yoshida. 0/1/all CSPs, half-integral A-path packing, and linear-time FPT algorithms. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 462–473. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00051.

21 Ashwin Jacob, Jari J. H. de Kroon, Diptapriyo Majumdar, and Venkatesh Raman. Deletion to scattered graph classes I – case of finite number of graph classes. *J. Comput. Syst. Sci.*, 138:103460, 2023. doi:10.1016/j.jcss.2023.05.005.

22 Ashwin Jacob, Diptapriyo Majumdar, and Venkatesh Raman. Parameterized complexity of deletion to scattered graph classes. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.18.

23 Ashwin Jacob, Diptapriyo Majumdar, and Venkatesh Raman. Deletion to scattered graph classes II – improved FPT algorithms for deletion to pairs of graph classes. *J. Comput. Syst. Sci.*, 136:280–301, 2023. doi:10.1016/j.jcss.2023.03.004.

24 Bart M. P. Jansen, Jari J. H. de Kroon, and Michał Włodarczyk. Single-exponential fpt algorithms for enumerating secluded $\mathcal{F}$-free subgraphs and deleting to scattered graph classes, 2023. arXiv:2309.11366.

25 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 160–169. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.53.

26 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Solving hard cut problems via flow-augmentation. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 149–168. SIAM, 2021. doi:10.1137/1.9781611976465.11.

27 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Directed flow-augmentation. 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 938–947. ACM, 2022. doi:10.1145/3519935.3520018.

28 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.

29 Daniel Lokshtanov and Dániel Marx. Clustering with local restrictions. *Inf. Comput.*, 222:278–292, 2013. doi:10.1016/j.ic.2012.10.016.

30 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.

31 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO model checking to highly connected graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 135:1–135:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.135.

32 Max-Jonathan Luckow and Till Fluschnik. On the computational complexity of length- and neighborhood-constrained path problems. *Inf. Process. Lett.*, 156:105913, 2020. doi:10.1016/j.ipl.2019.105913.

33 Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006. doi:10.1016/j.tcs.2005.10.007.

34 Dániel Marx. Important separators and parameterized algorithms. In Petr Kolman and Jan Kratochvíl, editors, *Graph-Theoretic Concepts in Computer Science – 37th International Workshop, WG 2011, Teplá Monastery, Czech Republic, June 21-24, 2011. Revised Papers*, volume 6986 of *Lecture Notes in Computer Science*, pages 5–10. Springer, 2011. doi:10.1007/978-3-642-25870-1_2.

35 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Trans. Algorithms*, 9(4):30:1–30:35, 2013. doi:10.1145/2500119.

36 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM J. Comput.*, 43(2):355–388, 2014. doi:10.1137/110855247.

37 Neeldhara Misra. Kernelization, Planar F-deletion. In *Encyclopedia of Algorithms*, pages 1033–1036. Springer, 2016. doi:10.1007/978-1-4939-2864-4_527.

38 Marcin Pilipczuk and Michal Ziobro. Experimental evaluation of parameterized algorithms for graph separation problems: Half-integral relaxations and matroid-based kernelization. *CoRR*, abs/1811.07779, 2018. arXiv:1811.07779.

39 Igor Razgon and Barry O'Sullivan. Almost 2-SAT is fixed-parameter tractable. *J. Comput. Syst. Sci.*, 75(8):435–450, 2009. doi:10.1016/j.jcss.2009.04.002.

40 A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.

41 Mingyu Xiao. Simple and improved parameterized algorithms for multiterminal cuts. *Theory Comput. Syst.*, 46(4):723–736, 2010. doi:10.1007/s00224-009-9215-5.

Is the Algorithmic Kadison-Singer Problem Hard?

Ben Jourdan
University of Edinburgh, UK

Peter Macgregor
University of Edinburgh, UK

He Sun
University of Edinburgh, UK

Abstract

We study the following $\mathsf{KS}_2(c)$ problem: let $c \in \mathbb{R}^+$ be some constant, and $v_1, \ldots, v_m \in \mathbb{R}^d$ be vectors such that $\|v_i\|^2 \leq \alpha$ for any $i \in [m]$ and $\sum_{i=1}^m \langle v_i, x\rangle^2 = 1$ for any $x \in \mathbb{R}^d$ with $\|x\| = 1$. The $\mathsf{KS}_2(c)$ problem asks to find some $S \subset [m]$, such that it holds for all $x \in \mathbb{R}^d$ with $\|x\| = 1$ that

$$\left|\sum_{i\in S}\langle v_i, x\rangle^2 - \frac{1}{2}\right| \leq c \cdot \sqrt{\alpha},$$

or report no if such S doesn't exist. Based on the work of Marcus et al. [15] and Weaver [20], the $\mathsf{KS}_2(c)$ problem can be seen as the algorithmic Kadison-Singer problem with parameter $c \in \mathbb{R}^+$.

Our first result is a randomised algorithm with one-sided error for the $\mathsf{KS}_2(c)$ problem such that (1) our algorithm finds a valid set $S \subset [m]$ with probability at least $1 - 2/d$, if such S exists, or (2) reports no with probability 1, if no valid sets exist. The algorithm has running time

$$O\left(\binom{m}{n} \cdot \mathrm{poly}(m, d)\right) \quad \text{for} \quad n = O\left(\frac{d}{\epsilon^2}\log(d)\log\left(\frac{1}{c\sqrt{\alpha}}\right)\right),$$

where ϵ is a parameter which controls the error of the algorithm. This presents the first algorithm for the Kadison-Singer problem whose running time is quasi-polynomial in m in a certain regime, although having exponential dependency on d. Moreover, it shows that the algorithmic Kadison-Singer problem is easier to solve in low dimensions. Our second result is on the computational complexity of the $\mathsf{KS}_2(c)$ problem. We show that the $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ problem is FNP-hard for general values of d, and solving the $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ problem is as hard as solving the $\mathsf{NAE\text{-}3SAT}$ problem.

2012 ACM Subject Classification Mathematics of computing → Probabilistic algorithms

Keywords and phrases Kadison-Singer problem, spectral sparsification

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.43

Related Version *Full Version*: https://arxiv.org/abs/2205.02161

Funding This work is supported by an EPSRC Early Career Fellowship (EP/T00729X/1).

1 Introduction

The Kadison-Singer problem [13] posed in 1959 asks whether every pure state on the (abelian) von Neumann algebra $\mathbb{D}$ of bounded diagonal operators on ℓ_2 has a unique extension to a pure state on $B(\ell_2)$, the von Neumann algebra of all bounded linear operators on the Hilbert space ℓ_2. The statement of the Kadison-Singer problem arises from work on the foundations of quantum mechanics done by Dirac in 1940s, and has been subsequently shown to be equivalent to numerous important problems in pure mathematics, applied mathematics, engineering and computer science [8]. Weaver [20] shows that the Kadison-Singer problem is equivalent to the following discrepancy question, which is originally posed as a conjecture.

▶ **Conjecture 1** (The KS_2 Conjecture). *There exist universal constants $\eta \geq 2$ and $\theta > 0$ such that the following holds. Let $v_1, \ldots, v_m \in \mathbb{C}^d$ satisfy $\|v_i\| \leq 1$ for all $i \in [m]$, and suppose $\sum_{i=1}^m |\langle u, v_i\rangle|^2 = \eta$ for every unit vector $u \in \mathbb{C}^d$. Then, there exists a partition S_1, S_2 of $[m]$ so that $\sum_{i \in S_j} |\langle u, v_i\rangle|^2 \leq \eta - \theta$, for every unit vector $u \in \mathbb{C}^d$ and every $j = \{1, 2\}$.*

As a major breakthrough in mathematics, Marcus, Spielman and Srivastava [15] prove that the KS_2 conjecture holds, and give an affirmative answer to the Kadison-Singer problem. Specifically, in this celebrated paper they show that, for any vectors $v_1, \ldots, v_m \in \mathbb{C}^d$ such that $\|v_i\|^2 \leq \alpha$ for any $i \in [m]$ and $\sum_{i=1}^m \langle v_i, x\rangle^2 = 1$ for any $x \in \mathbb{C}^d$ with $\|x\| = 1$, there is a partition S_1, S_2 of $[m]$ such that it holds for any $x \in \mathbb{C}^d$ with $\|x\| = 1$ and $j = 1, 2$ that $\left|\sum_{i \in S_j} \langle v_i, x\rangle^2 - 1/2\right| \leq 3 \cdot \sqrt{\alpha}$. The proof of this result is based on studying interlacing families of polynomials [14]. While analysing interlacing families of polynomials suffices to answer the KS_2 conjecture and, as a consequence, solve the Kadison-Singer problem, it is unclear if their existential proof on the partition guaranteed by the KS_2 conjecture can be turned into an efficient algorithmic construction; designing efficient algorithms for the Kadison-Singer problem is listed as a natural open question in [15]. This question is particularly interesting in theoretical computer science, since it is directly linked to constructing unweighted spectral sparsifiers [5] and spectrally thin trees [1], among many other applications in approximation algorithms. However, there has been little work on the algorithmic Kadison-Singer problem, and the complexity status of this problem is an important open question.

To address this question, we study the following KS_2 problem with some constant $c \in \mathbb{R}^+$:

▶ **Problem 2** (The $\mathsf{KS}_2(c)$ problem). *Given vectors $v_1, \ldots, v_m \in \mathbb{R}^d$ such that $\|v_i\|^2 \leq \alpha$ for any $i \in [m]$ and $\sum_{i=1}^m \langle v_i, x\rangle^2 = 1$ for any $x \in \mathbb{R}^d$ with $\|x\| = 1$, the $\mathsf{KS}_2(c)$ problem asks to*

- *find some $S \subset [m]$, such that it holds for all $x \in \mathbb{R}^d$ with $\|x\| = 1$ that*

$$\left|\sum_{i \in S} \langle v_i, x\rangle^2 - \frac{1}{2}\right| \leq c \cdot \sqrt{\alpha}, \tag{1}$$

- *or report no if such S doesn't exist.*

Notice that the KS_2 conjecture is equivalent to finding some subset $S \subset [m]$ as stated in Problem 2 for some constant c. Here we choose to formulate the discrepancy of any set $S \subset [m]$ in (1) as $c \cdot \sqrt{\alpha}$ for three reasons: first of all, Weaver [20] shows that the dependency on $O(\sqrt{\alpha})$ in (1) is tight, so the term $O(\sqrt{\alpha})$ is unavoidable when bounding the discrepancy; secondly, the KS_2 conjecture shows that the existence of any universal constant c in (1) suffices to prove the Kadison-Singer conjecture, and it is proven in [15] that the KS_2 conjecture holds for $c = 3$; however, studying the tightness of this constant remains an interesting open question on its own (Problem 8.1, [7]). Finally, as we will show shortly, the $\mathsf{KS}_2(c)$ problem belongs to different complexity classes with respect to different values of c, so introducing this parameter c allows us to better understand the complexity of the algorithmic Kadison-Singer problem.

1.1 Our Results

Our first result is an algorithm called RANDOMISED-KS$(\{v_i\}, c, \epsilon)$ for approximately solving the $\mathsf{KS}_2(c)$ problem for general values of c. For any constant c, $\epsilon < 1$, and any vectors $v_1, \ldots, v_m \in \mathbb{R}^d$ such that $\|v_i\|^2 \leq \alpha$ for all $i \in [m]$, we show that (i) if there exists an

S which satisfies (1), then with probability at least $(1 - 2/d)$ the algorithm returns a set $S' \subset \{v_i\}_{i=1}^m$ that satisfies

$$(1-\epsilon)\left(\frac{1}{2} - c\sqrt{\alpha}\right) \leq \sum_{v \in S'} \langle v, x\rangle^2 \leq (1+\epsilon)\left(\frac{1}{2} + c\sqrt{\alpha}\right) \tag{2}$$

for all unit vectors $x \in \mathbb{R}^d$, and (ii) if no set exists which satisfies (2), then with probability 1 the algorithm returns "no". Our result is summarised as follows:

▶ **Theorem 3.** *There is an algorithm,* RANDOMISED-$KS(\mathcal{I}, c, \epsilon)$*, such that for any instance* $\mathcal{I} \triangleq \{v_i\}_{i=1}^m$ *of the* $\mathsf{KS}_2(c)$ *problem with* $v_i \in \mathbb{R}^d$ *for* $d \geq 3$*, and for any* $\epsilon \in (0, 1)$*, the following holds:*

- *if there exists a set* $S \subset \mathcal{I}$ *such that*

$$\left(\frac{1}{2} - c\sqrt{\alpha}\right) \leq \sum_{v \in S} \langle v, x\rangle^2 \leq \left(\frac{1}{2} + c\sqrt{\alpha}\right)$$

  for all unit vectors $x \in \mathbb{R}^d$*, then with probability at least* $(1 - 2/d)$*, the* RANDOMISED-$KS(\mathcal{I}, c, \epsilon)$ *algorithm returns a subset* $S' \subset \mathcal{I}$ *which satisfies* (2) *for all unit vectors* $x \in \mathbb{R}^d$*.*
- *if there is no set* $S \subset \mathcal{I}$ *which satisfies* (2)*, then with probability* 1*, the* RANDOMISED-$KS(\mathcal{I}, c, \epsilon)$ *algorithm reports that no such set exists.*

The algorithm has running time

$$O\left(\binom{m}{n} \cdot \mathrm{poly}(m, d)\right) \quad \text{for} \quad n \triangleq O\left(\frac{d}{\epsilon^2}\log(d)\max\left(\log\left(\frac{1}{c\sqrt{\alpha}}\right), \log\left(\frac{1}{(1/2) - c\sqrt{\alpha}}\right)\right)\right).$$

▶ Remark 4. Since the most interesting instances of the $\mathsf{KS}_2(c)$ problem are the cases in which $1/2 + c\sqrt{\alpha}$ is bounded away from 1, we can assume that $c\sqrt{\alpha} \leq 1/2 - \sigma$ for some constant σ which implies that

$$n = O\left(\frac{d}{\epsilon^2}\log(d)\log\left(\frac{1}{c\sqrt{\alpha}}\right)\right).$$

Combining this with $d = \sum_{i=1}^m \|v_i\|^2 \leq \alpha m$, a constraint due to the isotropic nature of the input, shows that our algorithm runs in quasi-polynomial time in m when $d = O(\mathrm{polylog}(m))$.

Compared with the state-of-the-art that runs in $d^{O(m^{1/3}\alpha^{-1/4})}$ time [2], the most appealing fact of Theorem 3 is that it shows the $\mathsf{KS}_2(c)$ problem can be approximately solved in quasi-polynomial time when $d = O(\mathrm{poly}\log m)$. Moreover, for small values of c where a subset $S \subset [m]$ satisfying (1) isn't guaranteed to exist, our algorithm, with the same time complexity, is still able to find an S satisfying (2) with high probability if it exists, or report no with probability 1 otherwise. These two facts together show that both determining the existence of a valid subset S and finding such S are computationally much easier in low dimensions, regardless of the range of c. In addition, our result is much stronger than a random sampling based algorithm, which only works in the regime of $\alpha = O(1/\log d)$ [19], while our algorithm works even when there are vectors with much larger norm, e.g., $\alpha = \Theta(1)$. On the other side, like many optimisation problems that involve the dimension of input items in their formulation (e.g., multi-dimensional packing [10], and vector scheduling [4]), Theorem 3 indicates that the order of d might play a significant role in the hardness of the $\mathsf{KS}_2(c)$ problem, and the hard instances of the problem might be in the regime of $m = O(d)$.

Inspired by this, we study the computational complexity of the $\mathsf{KS}_2(c)$ problem for general values of d, where the number of input vectors satisfies $m = O(d)$. In order to study the "optimal" partitioning, for a given instance of the problem $\mathcal{I} = \{v_1, \ldots, v_m\}$, let

$$\mathcal{W}(\mathcal{I}) \triangleq \min_{S \subset \mathcal{I}} \max_{\substack{x \in \mathbb{R}^d \\ \|x\|=1}} \left| \sum_{v \in S} \langle v, x \rangle^2 - \frac{1}{2} \right|.$$

Then, we choose $c = 1/(4\sqrt{2})$ and notice that, for any vectors that satisfy the conditions of the $\mathsf{KS}_2(c)$ problem, there could be no subset S satisfying (1) for such c. As our second result, we prove that, for any $c \leq 1/(4\sqrt{2})$, distinguishing between instances for which $\mathcal{W}(\mathcal{I}) = 0$ and those for which $\mathcal{W}(\mathcal{I}) \geq c \cdot \sqrt{\alpha}$ is NP-hard. Our result is as follows:

▶ **Theorem 5.** *The* $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ *problem is* FNP*-hard for general values of* d*. Moreover, it is* NP*-hard to distinguish between instances of the* $\mathsf{KS}_2(c)$ *problem with* $\mathcal{W}(\mathcal{I}) = 0$ *from instances with* $\mathcal{W}(\mathcal{I}) \geq \left(1/4\sqrt{2}\right) \cdot \sqrt{\alpha}$*.*

▶ Remark 6. It's important to note that, when d is constant, the decision problem in Theorem 5 can be solved in polynomial time. For example, the 1-dimensional problem is equivalent to the PARTITION problem, in which we are given a set of real numbers $\mathcal{I} = \{x_1, \ldots, x_m\}$ such that $\sum_i x_i = 1$ and need to determine whether there is a subset $S \subset \mathcal{I}$ such that $\sum_{x \in S} x = 1/2$. In this setting,

$$\mathcal{W}(\mathcal{I}) = \min_{S \subset \mathcal{I}} \left| \left(\sum_{x \in S} x \right) - 1/2 \right|.$$

There is a well-known FPTAS for PARTITION which can distinguish between instances for which $\mathcal{W}(\mathcal{I}) = 0$ and those for which $\mathcal{W}(\mathcal{I}) \geq \epsilon$, for any $\epsilon > 0$. Theorem 5 implies that there is no such FPTAS for the optimisation version of the $\mathsf{KS}_2(c)$ problem for general d.

Theorem 5 shows that the isotropic structure of the $\mathsf{KS}_2(c)$ instance is not sufficient to make finding a partition easy when compared with similar problems. As such, the design of a potential polynomial-time algorithm for the Kadison-Singer problem would need to take some range of c into account and cannot solve the optimisation version of the $\mathsf{KS}_2(c)$ problem, otherwise one would end up solving an NP-hard problem. We remark that Theorem 5 shares the same style as the one for Spencer's Discrepancy Problem: given any input on N elements, Charikar et al. [9] shows that it is NP-hard to distinguish between the input with discrepancy zero and the one with discrepancy $\Omega(\sqrt{N})$, although it is known that a solution with $O(\sqrt{N})$ approximation can be computed efficiently [3].

1.2 Our Techniques

In this subsection we sketch our main techniques used in proving Theorems 3 and 5.

Proof Sketch of Theorem 3. We start by sketching the ideas behind our algorithmic result. First of all, it is easy to see that we can solve the $\mathsf{KS}_2(c)$ problem for any $c \in \mathbb{R}^+$ in $O\left(2^m \cdot \mathrm{poly}(m, d)\right)$ time, since we only need to enumerate all the 2^m subsets $S \subseteq \mathcal{I}$ of the input set $\mathcal{I}$ and check if every possible set S satisfies the condition (1). To express all the subsets of $\mathcal{I}$, we inductively construct level sets $\{\mathcal{L}_i\}_{i=0}^m$ with $\mathcal{L}_i \subseteq 2^{\mathcal{I}}$ as follows:

- initially, level $i = 0$ consists of a single set $\emptyset$, and we set $\mathcal{L}_0 = \{\emptyset\}$;
- based on $\mathcal{L}_{i-1}$ for any $1 \leq i \leq m$, we define $\mathcal{L}_i$ by $\mathcal{L}_i \triangleq \{S, S \cup \{v_i\} : S \in \mathcal{L}_{i-1}\}$.

It is important to see that, although $|\mathcal{L}_i|$ could be as high as 2^m, there are only m such level sets $\mathcal{L}_i$, which are constructed inductively in an *online* manner, and it holds for any $S \subseteq \mathcal{I}$ that $S \in \mathcal{L}_m$.

The bottleneck for improving the efficiency of this simple enumeration algorithm is the number of sets in $\mathcal{L}_m$, which could be exponential in m. To overcome this bottleneck, we introduce the notion of *spectral equivalence classes* to reduce $|\mathcal{L}_i|$ for any $i \in [m]$. Informally speaking, if there are different $S_1, S_2 \in \mathcal{L}_i$ for any $i \in [m]$ such that[1]

$$(1-\epsilon)\sum_{j\in S_2} v_j v_j^\intercal \preceq \sum_{j\in S_1} v_j v_j^\intercal \preceq (1+\epsilon)\sum_{j\in S_2} v_j v_j^\intercal$$

for some small ϵ, then we view S_1 and S_2 to be "spectrally equivalent" to each other[2]. It suffices to use one set to represent all of its spectral equivalences; hence, we only need to store the subsets which aren't spectrally equivalent to each other[3]. Since there is a spectral sparsifier of any S with $O(d\log(d)/\epsilon^2)$ vectors [11, 17], we can reduce the total number of stored subsets (i.e., the number of spectral equivalence classes) in $\mathcal{L}_i$ for any $i \in [m]$ to $\binom{m}{n}$ where $n = O\big(d\log(d)/\epsilon^2\big)$ which is no longer exponential in m.

Turning this idea into an algorithm design, we need be careful that the small approximation error introduced by every constructed spectral sparsifier does not compound as we construct sparsifiers from one level to another. In order to avoid this, we employ the online vector sparsification algorithm presented in [11]. This allows us to construct sparsifiers in $\mathcal{L}_i$ from the ones in $\mathcal{L}_{i-1}$ and the vector v_i. In addition, the construction in each level preserves the same approximation error as the previous one.

We highlight that the design of our algorithm for solving the $\mathsf{KS}_2(c)$ problem is entirely different from the previous work, which is based on analysing the properties of interlacing polynomials [2]. Moreover, one can view our use of online spectral sparsifiers in constructing spectral equivalence classes as an *encoding* strategy to reduce the enumeration space of the $\mathsf{KS}_2(c)$ problem. From this aspect, our work sheds light on potential applications of other tools well-studied in algorithmic spectral graph theory and numerical linear algebra, such as sparsification and sketching.

Proof Sketch of Theorem 5. Our proof of the FNP-hardness of the $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ problem is based on a reduction from the well-known NAE-3SAT problem [12] to a decision version of the $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ problem, which asks whether $\mathcal{W}(\mathcal{I}) = 0$ or $\mathcal{W}(\mathcal{I}) \geq \left(1/\left(4\sqrt{2}\right)\right)\sqrt{\alpha}$. Our overall reduction consists of two steps: we first build a reduction from the NAE-3SAT problem to the so-called NAE-3SAT-KS problem, and then build a reduction from the NAE-3SAT-KS problem to the $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ problem.

To sketch the first reduction, we examine the so-called NAE-3SAT-KS problem, which can be viewed as a restricted version of the NAE-3SAT problem, and used only as a tool to build the reduction from the NAE-3SAT problem to the $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ problem. Informally, the NAE-3SAT-KS problem consists of the 3SAT Boolean formula ψ, in which the number of occurrences of both u and $\bar{u}$ for every variable u in any ψ is limited with respect to some additional constraints and any two clauses of ψ share at most one literal; the NAE-3SAT-KS problem asks if there is a satisfying assignment for ψ such that every clause of ψ has at

[1] For any two matrices A and B of the same dimension, we write $A \preceq B$ if $B - A$ is positive semi-definite.
[2] Although this relationship is not symmetric, this informal definition is sufficient for the proof sketch and is not used directly in our analysis.
[3] The list of stored subsets can be thought of as an epsilon cover of all possible subsets.

least one true literal and at least one false literal; we refer the reader to Problem 11 in Section 3 for the formal definition of the NAE-3SAT-KS problem. Based on a reduction from the NAE-3SAT problem, we show that the NAE-3SAT-KS problem is NP-complete.

For the second and main reduction of our analysis, we build a reduction from the NAE-3SAT-KS problem to the $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ problem. Specifically, for an NAE-3SAT-KS instance ψ of n variables and m clauses, we construct a set A of $\Theta(n+m)$ vectors as a $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ instance, and each $v \in A$ has dimension $n+m$, such that the following properties hold:

- every vector v has norm $\|v\|^2 \leq 1/4$ and $\sum_{v\in A} vv^\intercal = I$;
- if ψ is a satisfiable instance of NAE-3SAT-KS, then there is a subset $S \subset A$ such that $\sum_{v\in S} vv^\intercal = (1/2)\cdot I$;
- if ψ is not a satisfiable instance of NAE-3SAT-KS, then for any subset $S \subset A$ there is always some $y \in \mathbb{R}^n$ with $\|y\| = 1$ such that $\left|\sum_{v\in S}\langle v, y\rangle^2 - 1/2\right| \geq 1/\left(8\sqrt{2}\right)$.

The key to proving these properties is the construction of a KS instance $\mathcal{I}$ from any formula ψ, and an analysis of the properties of $\sum_{v\in S} vv^\intercal$ for any $S \subseteq \mathcal{I}$ if ψ is an unsatisfiable instance of NAE-3SAT-KS. We think that such a reduction from any SAT instance to a KS instance is quite novel, and might be further employed to sharpen the constant $1/(4\sqrt{2})$.

1.3 Related Work

There has been little work on the algorithmic Kadison-Singer problem. Anari et al. [2] studies approximating the largest root of a real rooted polynomial and its applications to interlacing families, which are the main tool developed in [15] to prove the Kadison-Singer conjecture. They show that a valid partition promised by Weaver's KS_2 conjecture can be found in $d^{O(m^{1/3}\alpha^{-1/4})}$ time, suggesting that exhaustive search of all possibilities is not required for the algorithmic Kadison-Singer problem. Becchetti et al. [6] studies the algorithmic Kadison-Singer problem for graphs under some restricted condition. Specifically, they show that, if $G = (V, E)$ is an n-vertex and Δ-regular graph of $\Delta = \Omega(n)$ and the second eigenvalue of the adjacency matrix of G is at most a sufficient small constant times Δ, then an unweighted spectral sparsifier of G can be constructed efficiently.

Weaver [21] shows that the BSS-framework for constructing linear-sized spectral sparsifiers [5] can be adapted for the one-sided Kadison-Singer problem, where the term "one-sided" refers to the fact that the discrepancy of the algorithm's output can be only upper bounded.

Finally, independent of our work, Spielman and Zhang [18] studies the same complexity problem as ours. Different from our approach, their analysis starts with the $(3, 2\text{-}2)$ Set Splitting problem, which is a variant of the 2-2 Set Splitting problem. They prove that the $(3, 2\text{-}2)$ Set Splitting problem remains NP-hard even if no pair of sets intersects in more than one variable. Applying this, they show that the $\mathsf{KS}_2(c)$ problem is NP-hard for $c = 1/4$. While their result is slightly tighter than ours with respect to the value of c, the conclusions of the two works are essentially the same.

1.4 Notation

Let $[m] \triangleq \{1, \ldots, m\}$. For any integer j, we define vector $\mathbf{1}_j$, in which $\mathbf{1}_j(j) = 1$ and all of $\mathbf{1}_j$'s other entries are 0. For any integer $d \geq 1$, let $\mathbf{0}_{d\times d} \in \mathbb{R}^{d\times d}$ be the matrix in which every entry is equal to 0. We call a matrix A positive semi-definite (PSD) if $x^\intercal Ax \geq 0$ holds for any $x \in \mathbb{R}^d$. For any two matrices A and B, we write $A \preceq B$ if $B - A$ is PSD. The spectral norm of any matrix A is expressed by $\|A\|$.

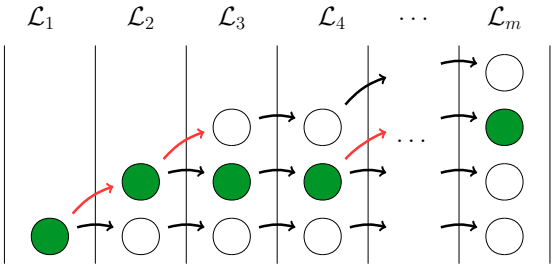


Figure 1 The construction of the sets $\mathcal{L}_i$ in Algorithm 1. Each $\mathcal{L}_{i-1}$ contains sparsifiers representing the spectral equivalence classes of the vectors $\{v_1, \ldots, v_{i-1}\}$. Then, $\mathcal{L}_i$ contains either one or two "children" of each sparsifier in $\mathcal{L}_{i-1}$, where the second child is added with some small probability which prevents $|\mathcal{L}_m|$ from growing exponentially with m. For a particular target subset $S \subseteq \{v_1, \ldots v_m\}$, there is some sequence of constructed sparsifiers which corresponds to the process of the online algorithm for constructing spectral sparsifiers [11], applied to S.

2 Algorithm Based on Spectral Equivalence Classes

This section discusses in detail the construction of spectral equivalence classes, and its application in designing a randomised algorithm for the $\mathsf{KS}_2(c)$ problem. We analyse the presented algorithm, and prove Theorem 3. All the proofs omitted from this section can be found in Appendix A.

2.1 Algorithm

Our algorithm consists of m iterations: in iteration i, the algorithm constructs the set $\mathcal{L}_i$ of spectral equivalence classes for the subsets $S \subseteq \{v_1, \ldots, v_i\}$. For each equivalence class, $\mathcal{L}_i$ contains a pair (S, B) where $S \subseteq \{v_1, \ldots v_i\}$ is a representative set in the equivalence class and $B \in \mathbb{R}^{d \times d}$ is a spectral sparsifier representing the equivalence class. Moreover, the algorithm constructs the representations of spectral equivalence classes in iteration i based on the ones maintained in iteration $i-1$. That is, instead of constructing all the subsets of $\{v_1, \ldots, v_i\}$ and grouping them into different spectral equivalence classes, the algorithm directly constructs the representations of the spectral equivalence classes of $\{v_1, \ldots, v_i\}$ based on its constructed equivalence classes of $\{v_1, \ldots, v_{i-1}\}$. This can be achieved by applying an online algorithm for constructing spectral sparsifiers, since, if we assume that in iteration $i-1$ every subset $S \subseteq \{v_1, \ldots, v_{i-1}\}$ is spectrally equivalent to some $(S', B') \in \mathcal{L}_{i-1}$ maintained by the algorithm, then both of S and $S \cup \{v_i\}$ are spectrally equivalent to S' and $S' \cup \{v_i\}$ in iteration i as well. As such, in iteration i we only need to ensure that the sets S' and $S' \cup \{v_i\}$ are still represented by some sparsifiers in $\mathcal{L}_i$.

Based on this, we can view all the vectors $v_1, \ldots, v_m$ as arriving *online* and, starting with the trivial spectral equivalence class defined by $\mathcal{L}_0 = \{(\emptyset, \mathbf{0}_{d \times d})\}$, the algorithm constructs the representations of spectral equivalence classes of $\{v_1, \ldots v_i\}$ in iteration i. Our algorithm applies the online algorithm for constructing spectral sparsifiers [11] (Lines 13-18 of Algorithm 1) to construct the representations of spectral equivalence classes of $\{v_1, \ldots, v_i\}$ based on those of $\{v_1, \ldots, v_{i-1}\}$. Since any subset of $\{v_1, \ldots v_m\}$ is spectrally equivalent to some set of vectors with size n where n is nearly linear in d [11], the number of spectral equivalence classes in any set $\mathcal{L}_i$ will be at most $\binom{m}{n}$. See Figure 1 for an illustration of the construction of the sets $\mathcal{L}_i$ and Algorithm 1 for the formal description of the algorithm.

▶ Remark 7. The if-condition on Line 10 of Algorithm 1 can be checked in polynomial time while introducing an arbitrarily small error, by constructing the matrix $\sum_{v \in S'} vv^\intercal$ and computing its eigenvalues.

Algorithm 1 RANDOMISED-KS($\mathcal{I} = \{v_i\}_{i=1}^m, c, \epsilon$), where $v_i \in \mathbb{R}^d$ and $\|v_i\|^2 \le \alpha$.

$\mu \leftarrow \epsilon/6$
$\lambda \leftarrow \min\left(c\sqrt{\alpha}, 1/2 - c\sqrt{\alpha}\right)$
$b \leftarrow 8\log(d)/\mu^2$
$n \leftarrow O\big(d\log(d)\log(1/\lambda)/\mu^2\big)$
$\mathcal{L}_0 \leftarrow \{(\emptyset, \mathbf{0}_{d\times d})\}$
for $i \leftarrow 1$ to m **do**
  $\mathcal{L}_i \leftarrow \emptyset$
  for $(S, B) \in \mathcal{L}_{i-1}$ *and* B *constructed with at most* n *vectors* **do**
    $S' \leftarrow S \cup \{v_i\}$
    if S' *satisfies* (2) **then**
      return S'
    end
    $p \leftarrow \min\left(b\,(1+\mu)\,v_i^\intercal\,(B+\lambda I)^{-1}\,v_i, 1\right)$
    if $X \le p$ *where* $X \sim \mathrm{Uniform}[0,1]$ **then**
      $B' \leftarrow B + \frac{1}{p}v_i v_i^\intercal$
      $\mathcal{L}_i \leftarrow \mathcal{L}_i \cup \{(S, B), (S', B')\}$
    else
      $\mathcal{L}_i \leftarrow \mathcal{L}_i \cup \{(S', B)\}$
    end
  end
end
return FAILURE

2.2 Analysis

First of all, notice that sparsifying $\sum_{v\in S} vv^\intercal$ for any $S \subseteq \mathcal{I}$ is equivalent to sparsifying the $|S| \times d$ matrix whose rows are defined by all the $v \in S$. Based on this, our proof uses the result from the online matrix sparsification algorithm [11] as a black box. Specifically, we apply the following lemma in our analysis, which is a special case of Theorem 2.3 from [11]. Notice that the algorithm described in Lemma 8 below corresponds to the sampling scheme used in Algorithm 1.

▶ **Lemma 8** ([11], Theorem 2.3). *Let S be a set of vectors $v_1, \ldots, v_m \in \mathbb{R}^d$, and let $A = \sum_{v\in S} vv^\intercal$. With $\mu, \delta \in [0,1]$, $b \triangleq 8\log(d)/\mu^2$ and $B_0 = \mathbf{0}_{d\times d}$, construct B_i inductively for $i \in [m]$ such that with probability*

$$p_i = \min\left(b(1+\mu)v_i^\intercal\left(B_{i-1} + \frac{\delta}{\mu}I\right)^{-1} v_i, 1\right),$$

we have

$$B_i = B_{i-1} + \frac{1}{p_i}v_i v_i^\intercal,$$

and with probability $1 - p_i$, we have $B_i = B_{i-1}$. Then, it holds with probability $(1 - 1/d)$ that

$$(1-\mu)A - \delta I \preceq B_m \preceq (1+\mu)A + \delta I,$$

and the number of vectors added to B_m is $O\left(d\log d\log\left(\mu\|A\|^2/\delta\right)/\mu^2\right)$.

Now, we analyse Algorithm 1. We begin by showing that, for each pair (S, B) constructed by Algorithm 1, B is a spectral sparsifier of S with high probability.

▶ **Lemma 9.** *Let $\mathcal{L}_i$ be the set constructed by Algorithm 1 at iteration i. Then, for any $(S^\star, B^\star) \in \mathcal{L}_i$, it holds with probability $(1 - 1/d)$ that*

$$(1-\mu)A_{S^\star} - \delta I \preceq B^\star \preceq (1+\mu)A_{S^\star} + \delta I$$

where $A_{S^\star} = \sum_{v \in S^\star} vv^\intercal$, and the parameters are set in Algorithm 1 to be $\mu = \epsilon/6$ and $\delta = \mu \min(c\sqrt{\alpha}, 1/2 - c\sqrt{\alpha})$.

Next we show that *any* set $S \subset \{v_1, \ldots, v_m\}$ is well approximated by one of the sparsifiers constructed in Algorithm 1.

▶ **Lemma 10.** *Let $\mathcal{I} = \{v_i\}_{i=1}^m$ be the input to Algorithm 1. Let $S \subseteq \mathcal{I}$ be any fixed set, and $A = \sum_{v \in S} vv^\intercal$. Then, with probability $(1 - 1/d)$, there is a matrix B constructed by Algorithm 1 such that*

$$(1-\mu)A - \delta I \preceq B \preceq (1+\mu)A + \delta I,$$

where $\mu = \epsilon/6$ and $\delta = \mu \min(c\sqrt{\alpha}, 1/2 - c\sqrt{\alpha})$.

Finally, to prove Theorem 3, we need only apply Lemma 10 for the target set $S \subset \mathcal{I}$, and Lemma 9 for one of the pairs (S', B) constructed by the algorithm. In particular, we do not need to take the union bound over all sparsifiers constructed by the algorithm; rather, it is sufficient that an accurate sparsifier is constructed for one specific target set.

Proof of Theorem 3. We first look at the case in which there is some $S \subset \mathcal{I}$, such that for $A_S = \sum_{i \in S} v_i v_i^\intercal$ it holds that $1/2 - c\sqrt{\alpha} \le x^\intercal A_S x \le 1/2 + c\sqrt{\alpha}$, for all unit vectors $x \in \mathbb{R}^d$. By Lemma 10, with probability greater than or equal to $1 - 1/d$, there exists some pair $(S', B) \in \mathcal{L}_m$ such that

$$(1-\mu)A_S - \delta I \preceq B \preceq (1+\mu)A_S + \delta I, \tag{3}$$

where $\mu = \epsilon/6$ and $\delta = \mu \min(c\sqrt{\alpha}, 1/2 - c\sqrt{\alpha}) \le \mu$. By Lemma 9, with probability $1 - 1/d$, we have $(1-\mu)A_{S'} - \delta I \preceq B \preceq (1+\mu)A_{S'} + \delta I$, where S' is the set constructed alongside B. Taking the union bound, with probability at least $1 - 2/d$, we have for any unit vector $x \in \mathbb{R}^d$ that

$$\begin{aligned} x^\intercal A_{S'} x &\le \frac{1+\mu}{1-\mu}\left(\frac{1}{2} + c\sqrt{\alpha}\right) + \frac{2\delta}{1-\mu} \\ &\le \frac{1+3\mu}{1-\mu}\left(\frac{1}{2} + c\sqrt{\alpha}\right) \\ &\le (1+\epsilon)\left(\frac{1}{2} + c\sqrt{\alpha}\right) \end{aligned} \qquad \text{and} \qquad \begin{aligned} x^\intercal A_{S'} x &\ge \frac{1-\mu}{1+\mu}\left(\frac{1}{2} - c\sqrt{\alpha}\right) - \frac{2\delta}{1-\mu} \\ &\ge \frac{1-3\mu}{1-\mu}\left(\frac{1}{2} - c\sqrt{\alpha}\right) \\ &\ge (1-\epsilon)\left(\frac{1}{2} - c\sqrt{\alpha}\right), \end{aligned}$$

where we use the definition of δ and the fact that $\epsilon = 6\mu \le 1$. Therefore, the set S' satisfies (2) and will be returned by Algorithm 1.

On the other side, notice that, by the condition on Line 10 of Algorithm 1, any set returned by the algorithm satisfies (2). Therefore, with probability 1 the algorithm will correctly report that there is no set $S \subset \mathcal{I}$ satisfying (2) if it is the case.

Finally, we analyse the running time of the algorithm. By Lemma 8, it holds that B is constructed from $O(n)$ vectors with probability at least $1 - 1/d$. For this reason, on Line 8 of Algorithm 1 we consider only the sparsifiers of size $O(n)$. The remaining part of the algorithm contributes only polynomial factors to its running time, so the total running time of the algorithm is $O\left(\binom{m}{n} \cdot \mathrm{poly}(m, d)\right)$. ◀

3 FNP-Hardness of $\mathsf{KS}_2\left(1/(4\sqrt{2})\right)$

This section studies the computational complexity of the $\mathsf{KS}_2(c)$ problem, and is organised as follows. In Section 3.1 we introduce the FNP complexity class. We formally define the NAE-3SAT-KS problem in Section 3.2, and prove that this problem is NP-hard. In Section 3.3, we build a reduction from the NAE-3SAT-KS problem to the $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ problem.

3.1 The FNP Complexity Class

In contrast with the complexity classes P and NP, the class FNP is used to study problems with output which is more complex than simply "yes" or "no". Formally, given a binary relation R and an input X, the corresponding *function problem* is to find Y such that $R(X,Y)$ holds or report "no" if no such Y exists. For example, we can take X to be an instance $\mathcal{I} = \{v_i\}_{i=1}^m$ of the $\mathsf{KS}_2(c)$ problem, and $Y \subseteq \mathcal{I}$ to be a candidate solution. Then, the relation $R_{\mathsf{KS}_2(c)}(\mathcal{I}, Y)$ holds if and only if Y satisfies (1). Any given binary relation R is in the class FNP iff there is a deterministic polynomial-time algorithm which can determine whether $R(X,Y)$ holds for a given pair (X,Y) [16]. Notice that every function problem has a natural corresponding decision problem. Specifically, given a binary relation R and a value of X, the decision problem asks whether there exists some Y such that $R(X,Y)$ holds. A function problem F is FNP-hard if there is a polynomial-time reduction from all problems in FNP to F. It is known that if the decision problem corresponding to F is NP-hard, then F is FNP-hard [16], and we will use this fact in our proof of Theorem 5.

3.2 NP-Completeness of NAE-3SAT-KS

In this subsection, we study the following NAE-3SAT-KS problem, and prove that the problem is NP-complete. We remark that we restrict ourselves to study SAT instances of a specific form here, as these SAT instances will be employed to prove the NP-hardness of the $\mathsf{KS}_2\left(1/(4\sqrt{2})\right)$ problem.

▶ **Problem 11** (NAE-3SAT-KS). *Given a 3SAT instance ψ consisting of a collection C of clauses over the set U of variables such that*

1. *every $c \in C$ has 3 literals,*
2. *for every $u \in U$, both of u and $\bar{u}$ appear in at most 2 clauses of C,*
3. *for every $u \in U$, at least one of u or $\bar{u}$ appears in exactly 2 clauses of C, and*
4. *any two clauses share at most one literal and no variable appears twice in the same clause,*

the NAE-3SAT-KS problem asks if there is a satisfying assignment for ψ such that every clause of ψ has at least one true literal and at least one false literal.

Our reduction is from the following well-known NP-complete problem.

▶ **Problem 12** (NAE-3SAT, [12]). *Given a 3SAT instance ψ that consists of a collection C of clauses over the set U of variables such that every clause $c \in C$ has 3 literals, the NAE-3SAT problem asks if there is a satisfying assignment for ψ such that every clause of ψ has at least one true literal and at least one false literal.*

▶ **Theorem 13.** *The NAE-3SAT-KS problem is NP-complete.*

Proof. Given any NAE-3SAT-KS instance ψ and an assignment to ψ's variables, it's straightforward to check in polynomial time if this is a satisfying assignment, and every clause of ψ has at least one true literal and at least one false literal. Hence, the NAE-3SAT-KS problem is in NP.

To prove that the NAE-3SAT-KS problem is NP-complete, we build a reduction from the NAE-3SAT problem to the NAE-3SAT-KS problem. Specifically, for any NAE-3SAT instance (U, C), where U is the set of variables and C is a collection of clauses, we construct an NAE-3SAT-KS instance (U', C') such that (U, C) is satisfiable in NAE-3SAT if and only if (U', C') is satisfiable in NAE-3SAT-KS. Our construction of (U', C') is as follows. Initially, we set $U' = U$ and $C' = C$. Then, for any variable x which appears only once in C, we remove x from U' and the corresponding clause from C' since the clause can always be satisfied by setting x appropriately and so removing the clause does not change the satisfiability of (U', C'). Then, for every remaining variable x, we replace the instances of x and $\bar{x}$ with new variables and add additional clauses to ensure that the satisfiability is unchanged. Specifically, for each x left in U' let $n_1 = |\{c \in C : x \in c\}|$, $n_2 = |\{c \in C : \bar{x} \in c\}|$, and set $n = n_1 + n_2$. Then, we introduce new variables $x_1, \ldots, x_n$ and replace the instances of x in C' with $x_1, \ldots, x_{n_1}$. Similarly, we replace the instances of $\bar{x}$ with $\bar{x}_{n_1+1}, \ldots, \bar{x}_n$.

Now, in order to ensure that (U', C') is satisfiable if and only if (U, C) is satisfiable, we introduce new clauses to C' which have the effect of constraining the variables $x_1, \ldots, x_n$ to have the same truth value in any satisfying assignment. To achieve this, let $n' \geq n$ be an odd number, and we introduce additional new variables $y_1, \ldots, y_{n'}$ and clauses

$$(\bar{y}_i \vee \bar{y}_{i+1} \vee y_{i+2}) \quad \text{for any } i \in [1, n'], \tag{4}$$

where the indices are taken modulo n'. We will see that these clauses ensure that the y_i variables must all have the same value in a satisfying assignment. We see this by a simple case distinction.

- Case 1: $y_1 = y_2$ in a satisfying assignment. Then, by the first clause in (4) it must be that $y_2 = y_3$ since there must be at least one true literal and one false literal in each satisfied clause. Proceeding inductively through the clauses in (4), we establish that $y_1 = y_2 = \ldots = y_{n'}$.
- Case 2: $y_1 \neq y_2$ in a satisfying assignment. We will show that this leads to a contradiction. By the last clause in (4), $y_{n'} \neq y_1$ since there must be at least one true literal and one false literal. Again, we proceed inductively from the $(n'-1)$th clause in (4) down to establish that $y_1 \neq y_2, y_2 \neq y_3, \ldots, y_{n'-1} \neq y_{n'}$. As such, we have $y_1 = y_3 = \ldots = y_{2i+1}$ which is a contradition since n' is odd and we have already established that $y_1 \neq y_{n'}$.

As such, we can use the variables $y_1, \ldots, y_{n'}$ with the assumption that they have the same value in any satisfying assignment of (U', C'). It remains to construct clauses to guarantee that the variables $x_1, \ldots, x_n$ have the same value in any satisfying assignment. We add the clauses

$$(x_i \vee \bar{x}_{i+1} \vee y_i) \quad \text{for any } i \in [1, n], \tag{5}$$

where the indices are taken modulo n. We will show that $x_1 = x_2 = \ldots = x_n$ in a satisfying assignment by case distinction.

- Case 1: $x_1 = y_i$ for all i. By the first clause in (5), it must be that $x_1 = x_2$ since we cannot have $x_1 = \bar{x}_2 = y_i$ in a satisfying assignment. Then, proceeding inductively using each clause in turn we establish that $x_1 = x_2 = \ldots = x_n$.
- Case 2: $\bar{x}_1 = y_i$ for all i. By the last clause in (5), it must be that $\bar{x}_n = \bar{x}_1$ since we cannot have $x_n = \bar{x}_1 = y_n$ in a satisfying assignment. Then, proceeding inductively from the $(n-1)$th clause down, we establish that $\bar{x}_1 = \bar{x}_2 = \ldots = \bar{x}_n$.

Notice that by this construction, each literal x_i, $\bar{x}_i$, y_i, and $\bar{y}_i$ now appears at most twice in C', no two clauses share more than one literal and no literal appears twice in the same clause. Additionally, every x_i and $\bar{x}_i$ appears exactly once in the clauses added by (5).

Since the variable x_i also appears exactly once in the clauses corresponding directly to C, requirement (3) of the NAE-3SAT-KS problem is satisfied. Moreover, we have that (U', C') has a satisfying assignment if (U, C) has a satisfying assignment; this follows by setting the values of $x_1, \ldots, x_n$ in U' to the value of their corresponding $x \in U$. On the other hand, any satisfying assignment of (U', C') corresponds to a satisfying assignment of (U, C), since we must have that $x_1 = \ldots = x_n$ and can set the value of $x \in U$ to be the same value to get a satisfying assignment of (U, C). Finally, notice that our new instance (U', C') of NAE-3SAT-KS can be constructed in polynomial time in the size of the instance (U, C) of NAE-3SAT. This completes the proof. ◀

3.3 FNP-Hardness of $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$

We now show that the $\mathsf{KS}_2(c)$ problem is FNP-hard for any $c \leq 1/(4\sqrt{2})$, i.e., Theorem 5. At a high level, our proof is by reduction from the NAE-3SAT-KS problem. Given an instance of the NAE-3SAT-KS problem, we will construct an instance $\mathcal{I}$ of $\mathsf{KS}_2(c)$ such that

1. if the NAE-3SAT-KS instance is satisfiable, then there is a set $S \subset \mathcal{I}$ with $\sum_{v \in S} vv^\intercal = (1/2) \cdot I$, and
2. if the NAE-3SAT-KS instance is not satisfiable, then for all sets $S \subset \mathcal{I}$ we have

$$\left\| \sum_{v \in S} vv^\intercal - \frac{1}{2} I \right\| \geq \frac{1}{4\sqrt{2}} \cdot \sqrt{\alpha}.$$

This will establish that the $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ problem is FNP-complete, and that it is NP-hard to distinguish between instances of $\mathsf{KS}_2(c)$ with $\mathcal{W}(\mathcal{I}) = 0$ and those for which $\mathcal{W}(\mathcal{I}) \geq \left(1/4\sqrt{2}\right)\sqrt{\alpha}$.

Proof of Theorem 5. We prove that $\mathsf{KS}_2(1/(4\sqrt{2}))$ is NP-hard by a reduction from the NAE-3SAT-KS problem to the decision version of the $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ problem. We are given an instance (U, C) of the NAE-3SAT-KS problem, and construct an instance of $\mathsf{KS}_2(c)$. Let us refer to

- the clauses in C as $c_1, \ldots c_m$;
- the variables in U as $x_1, \ldots, x_n$; we sometimes write x_i and $\bar{x}_i$ for the un-negated and negated literals.

Our constructed $\mathsf{KS}_2(c)$ instance has $O(n+m)$ dimensions. Specifically, there is one dimension for each clause in C and one dimension for each variable in U which appears both negated and un-negated in C. We use d_j^c to refer to the dimension corresponding to clause c_j, and d_j^x to refer to the dimension corresponding to variable x_j. We add $O(m+n)$ vectors to our $\mathsf{KS}_2(c)$ instance. Conceptually, we add one vector for each clause and 4 vectors for each literal. We use v_j^c to refer to the vector corresponding to clause c_j, and $v_{j,1}^x$ to $v_{j,4}^x$ or $v_{j,1}^{\bar{x}}$ to $v_{j,4}^{\bar{x}}$ to refer to the vectors corresponding to the literal x_j or $\bar{x}_j$. For each clause c_j, we set $v_j^c(d_j^c) = 1/2$, and set the other entries of v_j^c to be 0. Table 1 completes the definition of the vectors corresponding to literals. For each literal, we define only the value on the dimensions corresponding to the variable and the clauses containing the literal; all other entries in the vector are 0. Let A be the set of vectors defined above. Notice that the squared norms of the vectors in A are bounded above by $1/4$ and so $\alpha = 1/4$ in the constructed $\mathsf{KS}_2(c)$ instance.

To complete the reduction, we'll show the following:

1. It holds that $\sum_{v \in A} vv^\intercal = I$.
2. If the original NAE-3SAT-KS instance has a satisfying assignment, then there's a set $S \subset A$ such that $\sum_{v \in S} vv^\intercal = \frac{1}{2} \cdot I$.

Table 1 The construction of the vectors in the $\mathsf{KS}_2(c)$ instance for a literal x_i appearing in clause c_j and possibly also in c_k. If a literal appears in only one clause, c_j, we ignore the middle column corresponding to c_k; i.e., the vectors corresponding to x_i are non-zero only on dimensions d_j^c and d_i^x.

Vector	Value on d_j^c	Value on d_k^c	Value on d_i^x
$v_{i,1}^x$	$1/4$	$1/4$	$1/\sqrt{8}$
$v_{i,2}^x$	$1/4$	$1/4$	$-1/\sqrt{8}$
$v_{i,3}^x$	$1/4$	$-1/4$	$1/\sqrt{8}$
$v_{i,4}^x$	$1/4$	$-1/4$	$-1/\sqrt{8}$

3. Any set $S \subset A$ with

$$\left\| \sum_{v \in S} vv^\intercal - \frac{1}{2} I \right\| < \frac{1}{8\sqrt{2}} = \frac{1}{4\sqrt{2}} \sqrt{\alpha}$$

corresponds to a satisfying assignment of the original NAE-3SAT-KS instance.

Vectors in A are isotropic. Let $B = \sum_{v \in A} vv^\intercal$. Then, for any variable x_i, we have that

$$B(d_i^x, d_i^x) = \sum_{v \in A} v(d_i^x)^2 = \sum_{j=1}^{4} v_{i,j}^x (d_i^x)^2 + \sum_{j=1}^{4} v_{i,j}^{\bar{x}} (d_i^x)^2 = 1.$$

Additionally, for any clause c_i we have that

$$B(d_i^c, d_i^c) = \sum_{v \in A} v(d_i^c)^2 = v_i^c (d_i^c)^2 + \sum_{x_j \in c} \sum_{k=1}^{4} v_{j,k}^x (d_i^c)^2 = \frac{1}{4} + 3 \cdot \frac{1}{4} = 1.$$

This demonstrates that the diagonal entries of B are all 1. We now see that the off-diagonal entries are all 0. First, notice that for any two dimensions relating to variables, d_i^x and d_j^x, we have

$$B(d_i^x, d_j^x) = \sum_{v \in A} v(d_i^x) v(d_j^x) = 0,$$

since there is no vector in A with a non-zero contribution to more than one dimension corresponding to a variable. Now, let us consider two dimensions corresponding to different clauses c_i and c_j. We have

$$B(d_i^c, d_j^c) = \sum_{v \in A} v(d_i^c) v(d_j^c) = \sum_{x_k \in c_i \cap c_j} \sum_{\ell=1}^{4} v_{k,\ell}^x (d_i^c) \cdot v_{k,\ell}^x (d_j^c) = 0,$$

where we use the fact that c_i and c_j share at most one literal. Finally, consider the case when one dimension corresponds to the clause c_i and the other dimension corresponds to the variable x_j. If the variable x_j does not appear in c_i, then there are no vectors with a non-zero contribution to the two dimensions and so the entry is 0. Otherwise, we have

$$B(d_i^c, d_j^x) = \sum_{v \in A} v(d_i^c) v(d_j^x) = \sum_{k=1}^{4} v_{i,k}^x (d_i^c) v_{i,k}^x (d_j^x) = \frac{1}{4\sqrt{8}} + \frac{1}{4\sqrt{8}} - \frac{1}{4\sqrt{8}} - \frac{1}{4\sqrt{8}} = 0,$$

where we use the fact that no variable appears twice in the same clause. This completes the proof that $\sum_{v \in A} vv^\intercal = I$.

If the NAE-3SAT-KS instance is satisfiable, then there is a solution to $\mathsf{KS}_2(1/(4\sqrt{2}))$. Given a satisfying assignment to NAE-3SAT-KS, let $T \subset U$ be the set of variables which are set to be TRUE and let $F \subset U$ be the set of variables which are set to be FALSE. Recall that in a satisfying assignment, each clause in C contains either 1 or 2 true literals. Let $C' \subset C$ be the set of clauses with exactly 1 true literal in the satisfying assignment. Then, we define S to be

$$S \triangleq \{v^x_{i,1}, v^x_{i,2}, v^x_{i,3}, v^x_{i,4} : x_i \in T\} \cup \{v^{\bar{x}}_{i,1}, v^{\bar{x}}_{i,2}, v^{\bar{x}}_{i,3}, v^{\bar{x}}_{i,4} : x_i \in F\} \cup \{v^c_i : c_i \in C'\},$$

and we show that $\sum_{v\in S} vv^\intercal = \frac{1}{2}I$. We can repeat the previous calculations, this time setting $B = \sum_{v\in S} vv^\intercal$ to show that $B = (1/2)I$. Specifically, for any variable x_i, it holds that $B(d^x_i, d^x_i) = \frac{1}{2}$ since only the vectors corresponding to the negated *or* un-negated variable are included. For any clause $c_i \in C'$, we have

$$B(d^c_i, d^c_i) = v^c_i(d^c_i)^2 + \sum_{k=1}^{4} v^x_{j,k}(d^c_i)^2 = 1/4 + 1/4 = 1/2,$$

where x_j is the literal which is set to be true in the clause c_i. Similarly, for any clause in $c_i \in C \setminus C'$, we have

$$B(d^c_i, d^c_i) = \sum_{k=1}^{4} v^x_{j,k}(d^c_i)^2 + \sum_{k=1}^{4} v^x_{\ell,k}(d^c_i)^2 = 1/4 + 1/4 = 1/2,$$

where the literals x_j and x_ℓ are set to be true in the clause c_i. Then, notice that the calculations for the off-diagonal entries follow in the same way as before. This completes the proof that a satisfying assignment for NAE-3SAT-KS implies a solution to the $\mathsf{KS}_2(1/(4\sqrt{2}))$ problem.

If there is a solution to $\mathsf{KS}_2(c)$, then the NAE-3SAT-KS instance is satisfiable. We prove this by a contrapositive argument. That is, we show that for any set S' which does not correspond to a satisfying assignment of the NAE-3SAT-KS problem, there must be some vector y with $\|y\| = 1$ such that

$$\left| y^\intercal \left(\sum_{v\in S'} vv^\intercal \right) y - \frac{1}{2} \right| \geq \epsilon \tag{6}$$

for $\epsilon = \frac{1}{8\sqrt{2}}$. Specifically, we will analyse three cases, and show that

1. if there is some variable x_i such that S' does not contain exactly 4 of the vectors

   $$\{v^x_{i,1}, v^x_{i,2}, v^x_{i,3}, v^x_{i,4}, v^{\bar{x}}_{i,1}, v^{\bar{x}}_{i,2}, v^{\bar{x}}_{i,3}, v^{\bar{x}}_{i,4}\},$$

   then there is a vector y satisfying (6) for $\epsilon = 1/8$;
2. if Case (1) does not apply, then if there is some literal x_i such that S' contains 1, 2, or 3 of the vectors $\{v^x_{i,1}, v^x_{i,2}, v^x_{i,3}, v^x_{i,4}\}$, then there is a vector y satisfying (6) for $\epsilon = 1/(8\sqrt{2})$;
3. if neither Case (1) nor (2) applies, then if S' does not correspond to a satisfying assignment of the original NAE-3SAT-KS instance, there must be a vector y satisfying (6) for $\epsilon = 1/4$.

For the first case, suppose that there is some variable x_i such that S' does not contain exactly 4 vectors corresponding to the variable x_i. Let $k \neq 4$ be the number of such vectors, and let y be the vector with all zeros except for $y(d^x_i) = 1$. Notice that

$$\left| y^\intercal \left(\sum_{v\in S'} vv^\intercal \right) y - \frac{1}{2} \right| = \left| \sum_{v\in S'} v(d^x_i)^2 - \frac{1}{2} \right| = \left| \frac{k}{8} - \frac{1}{2} \right| \geq \frac{1}{8}.$$

Table 2 The absolute values of off-diagonal entries in $B = \sum_{v \in S'} vv^\intercal$, based on which vectors corresponding to the literal x_i are included in S'. We assume x_i appears in the clauses c_j and c_k.

Vectors in S'	$\lvert B(d_i^x, d_j^c)\rvert$	$\lvert B(d_i^x, d_k^c)\rvert$	$\lvert B(d_j^c, d_k^c)\rvert$
One vector $v_{i,\ell}^x$	$1/\left(8\sqrt{2}\right)$	$1/\left(8\sqrt{2}\right)$	$1/16$
Vectors $v_{i,1}^x$ and $v_{i,2}^x$	0	0	$1/8$
Vectors $v_{i,1}^x$ and $v_{i,3}^x$	$1/\left(4\sqrt{2}\right)$	0	0
Vectors $v_{i,1}^x$ and $v_{i,4}^x$	0	$1/\left(4\sqrt{2}\right)$	0
Vectors $v_{i,2}^x$ and $v_{i,3}^x$	0	$1/\left(4\sqrt{2}\right)$	0
Vectors $v_{i,2}^x$ and $v_{i,4}^x$	$1/\left(4\sqrt{2}\right)$	0	0
Vectors $v_{i,3}^x$ and $v_{i,4}^x$	0	0	$1/8$
Three vectors $v_{i,\ell}^x$	$1/\left(8\sqrt{2}\right)$	$1/\left(8\sqrt{2}\right)$	$1/16$

For the second case, suppose that the set S' contains 4 vectors for each variable, but there is some literal x_i such that S' contains some but not all of the vectors corresponding to x_i. By Condition 3 of the NAE-3SAT-KS problem (Problem 11) we can assume that x_i appears in two clauses c_j and c_k. Otherwise, this is the case for $\bar{x}_i$ and S' contains some, but not all, of the vectors corresponding to $\bar{x}_i$ since it contains exactly 4 vectors corresponding to the variable x_i. Now, we define $B = \sum_{v \in S'} vv^\intercal$ and we consider the absolute values of certain off-diagonal entries in B, which are summarised in Table 2. Notice that, regardless of which vectors corresponding to x_i are included, there are two indices $\hat{d}_1$ and $\hat{d}_2$ such that $\left|B(\hat{d}_1, \hat{d}_2)\right| \geq \frac{1}{8\sqrt{2}}$. Using the indices $\hat{d}_1$ and $\hat{d}_2$, define the unit vector

$$y = \begin{cases} \frac{1}{\sqrt{2}}(\mathbf{1}_{\hat{d}_1} + \mathbf{1}_{\hat{d}_2}) & \text{if } \operatorname{sgn}(B(\hat{d}_1, \hat{d}_1) + B(\hat{d}_2, \hat{d}_2) - 1) = \operatorname{sgn}(B(\hat{d}_1, \hat{d}_2)) \\ \frac{1}{\sqrt{2}}(\mathbf{1}_{\hat{d}_1} - \mathbf{1}_{\hat{d}_2}) & \text{otherwise} \end{cases}$$

where $\operatorname{sgn}(\cdot)$ is the sign function. Then we have

$$\begin{aligned} \left|y^\intercal B y - \frac{1}{2}\right| &= \left|\frac{1}{2}\Big(B(\hat{d}_1, \hat{d}_1) + B(\hat{d}_2, \hat{d}_2) \pm B(\hat{d}_1, \hat{d}_2) \pm B(\hat{d}_2, \hat{d}_1)\Big) - \frac{1}{2}\right| \\ &= \frac{1}{2}\left|B(\hat{d}_1, \hat{d}_1) + B(\hat{d}_2, \hat{d}_2) - 1 \pm 2B(\hat{d}_1, \hat{d}_2)\right| \\ &= \frac{1}{2}\left(\left|B(\hat{d}_1, \hat{d}_1) + B(\hat{d}_2, \hat{d}_2) - 1\right| + 2\left|B(\hat{d}_1, \hat{d}_2)\right|\right) \\ &\geq \left|B\left(\hat{d}_1, \hat{d}_2\right)\right| \\ &\geq \frac{1}{8\sqrt{2}}, \end{aligned}$$

where the third equality follows by the construction of y.

Finally, we consider the third case, in which there are 4 vectors in S' for each variable, and all 4 vectors correspond to the same literal. It is clear that such a set S' corresponds unambiguously to an assignment for the original variables in the NAE-3SAT-KS instance: specifically, one can set a variable x_i to be TRUE if S' contains $\{v_{i,1}^x, v_{i,2}^x, v_{i,3}^x, v_{i,4}^x\}$, and set x_i to be FALSE if S' contains $\{v_{i,1}^{\bar{x}}, v_{i,2}^{\bar{x}}, v_{i,3}^{\bar{x}}, v_{i,4}^{\bar{x}}\}$. Then, suppose that there is some clause $c_j \in C$ which is not satisfied by this assignment. This implies that either all 12 of the vectors corresponding to literals in c_j are included in S', or none of the vectors corresponding to literals in c_j are included in S'. In either case, we can set y to be the indicator vector of the dimension d_j^c, and have that

$$|y^\intercal By - 1/2| = \left| \sum_{v \in S'} v(d_j^c)^2 - 1/2 \right| \geq 1/4$$

since we can either include v_j^c or not in order to set $\sum_{v \in S'} v(d_j^c)^2$ equal to either $1/4$ or $3/4$.

This completes the reduction from the NAE-3SAT-KS problem to the decision version of the $\mathsf{KS}_2(c)$ problem for $c \leq 1/(4\sqrt{2})$, which implies that $\mathsf{KS}_2\left(1/\left(4\sqrt{2}\right)\right)$ is FNP-hard. Furthermore, notice that by the reduction in this proof,

- if the NAE-3SAT-KS instance is satisfiable, then the constructed instance $\mathcal{I}$ of the $\mathsf{KS}_2(c)$ problem satisfies $\mathcal{W}(\mathcal{I}) = 0$, and
- if the NAE-3SAT-KS instance is not satisfiable, then the constructed instance $\mathcal{I}$ of the $\mathsf{KS}_2(c)$ problem satisfies $\mathcal{W}(\mathcal{I}) \geq 1/\left(4\sqrt{2}\right) \cdot \sqrt{\alpha}$.

This shows that distinguishing between instances with $\mathcal{W}(\mathcal{I}) = 0$ and $\mathcal{W}(\mathcal{I}) \geq 1/\left(4\sqrt{2}\right) \cdot \sqrt{\alpha}$ is NP-hard, and completes the proof. ◀

4 Conclusion

This paper studies the algorithms and complexity of the Kadison-Singer problem through the $\mathsf{KS}_2(c)$ problem, and presents two results. On one side, we prove that the $\mathsf{KS}_2(c)$ problem for any $c \in \mathbb{R}^+$ can be solved in quasi-polynomial time when $d = O(\log m)$, which suggests that the problem is much easier to solve in low dimensions. The key to our algorithm design is a novel application of online spectral sparsification subroutines, with which we are able to efficiently construct representations of all spectral equivalence classes over time and reduce the enumeration space of the candidate solutions. We expect that our work could motivate more research on the applications of spectral sparsification and related problems in numerical linear algebra to the algorithmic Kadison-Singer problem.

On the other side, our NP-hardness result shows that the Kadison-Singer type problem for arbitrary dimensions can be as hard as solving the SAT problem, and the $\mathsf{KS}_2(c)$ problem belongs to different complexity classes for different values of c. Hence, more refined studies on the classification of its computational complexity would help us better understand the complexity of the algorithmic Kadison-Singer problem. In our point of view, both directions left from the paper are very interesting, and we leave these for future work.

References

1. Nima Anari and Shayan Oveis Gharan. The Kadison-Singer problem for strongly Rayleigh measures and applications to asymmetric TSP. *CoRR*, abs/1412.1143, 2014. `arXiv:1412.1143`.
2. Nima Anari, Shayan Oveis Gharan, Amin Saberi, and Nikhil Srivastava. Approximating the largest root and applications to interlacing families. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 1015–1028, 2018.
3. Nikhil Bansal. Constructive algorithms for discrepancy minimization. In *51th Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 3–10, 2010.
4. Nikhil Bansal, Tim Oosterwijk, Tjark Vredeveld, and Ruben van der Zwaan. Approximating vector scheduling: Almost matching upper and lower bounds. *Algorithmica*, 76(4):1077–1096, 2016.
5. Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.
6. Luca Becchetti, Andrea E. F. Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Finding a bounded-degree expander inside a dense one. In *31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, pages 1320–1336, 2020.

7 Peter G. Casazza. Consequences of the Marcus/Spielman/Stivastava solution to the Kadison-Singer problem. *CoRR*, abs/1407.4768, 2014. arXiv:1407.4768.

8 Peter G Casazza, Matthew Fickus, Janet C Tremain, and Eric Weber. The Kadison-Singer problem in mathematics and engineering: a detailed account. *Contemporary Mathematics*, 414:299, 2006.

9 Moses Charikar, Alantha Newman, and Aleksandar Nikolov. Tight hardness results for minimizing discrepancy. In *22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'11)*, pages 1607–1614, 2011.

10 Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM Journal on Computing*, 33(4):837–851, 2004.

11 Michael B. Cohen, Cameron Musco, and Jakub Pachocki. Online row sampling. *Theory of Computing*, 16(15):1–25, 2020.

12 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

13 Richard Kadison and Isadore Singer. Extensions of pure states. *American Journal of Mathematics*, 81:383–400, 1959.

14 Adam Marcus, Daniel A. Spielman, and Nikhil Srivastava. Interlacing families I: bipartite Ramanujan graphs of all degrees. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS'13)*, pages 529–537, 2013.

15 Adam W. Marcus, Daniel A. Spielman, and Nikhil Srivastava. Interlacing families II: mixed characteristic polynomials and the Kadison-Singer problem. *Annals of Mathematics*, 182(1):327–350, 2015.

16 Elaine Rich. *Automata, computability and complexity: theory and applications*. Pearson Prentice Hall Upper Saddle River, 2008.

17 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

18 Daniel A. Spielman and Peng Zhang. Hardness results for Weaver's discrepancy problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM '22)*, pages 40:1–40:14, 2022.

19 Joel A Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of computational mathematics*, 12(4):389–434, 2012.

20 Nicholas Weaver. The Kadison-Singer problem in discrepancy theory. *Discrete Mathematics*, 278(1-3):227–239, 2004.

21 Nik Weaver. The Kadison-Singer problem in discrepancy theory, II. *CoRR*, abs/1303.2405, 2013. arXiv:1303.2405.

A Omitted Proofs from Section 2

In this section we present the proofs omitted from Section 2.

Proof of Lemma 9. We will show that for any pair $(S^\star, B^\star)$ constructed by Algorithm 1, $B^\star$ is equivalent to the output of the algorithm described in Lemma 8 when applied to $S^\star$. We prove this by induction on i. The base case $i = 0$ follows immediately from the initialisation of $\mathcal{L}_0 = \{(\emptyset, \mathbf{0}_{d\times d})\}$. For the inductive step we show that the conclusion holds for every pair in $\mathcal{L}_i$, assuming it holds for every pair in $\mathcal{L}_{i-1}$. For each pair $(S^\star, B^\star) \in \mathcal{L}_i$, the proof proceeds by a case distinction.

Case 1: $(S^\star, B^\star) \in \mathcal{L}_{i-1}$. This case corresponds to the pairs (S, B) added on Line 16 of Algorithm 1. Accordingly, by the inductive hypothesis, we have that $B^\star$ is equivalent to the output of the algorithm described in Lemma 8 applied to $S^\star$.

Case 2: $(S^\star, B^\star) \notin \mathcal{L}_{i-1}$**.** This case covers the pairs involving S' added on Lines 16 and 18 of Algorithm 1. Let (S, B_{i-1}) be the pair in $\mathcal{L}_{i-1}$ from which $(S^\star, B^\star)$ is constructed. Notice that $S^\star = S \cup \{v_i\}$. Then, by the construction of Algorithm 1, with probability p_i, we have

$$B^\star = B_{i-1} + \frac{1}{p_i} v_i v_i^\intercal$$

and with probability $1 - p_i$, we have $B^\star = B_{i-1}$, where p_i is the probability defined in Lemma 8. As such, $B^\star$ is the result of applying an iteration of the algorithm defined in Lemma 8, for the new vector v_i. This maintains that $B^\star$ is equivalent to the output of the Lemma 8 algorithm applied to $S^\star$ and completes the inductive argument. ◀

Proof of Lemma 10. We prove that one of the matrices B constructed by Algorithm 1 is equivalent to the output of the algorithm defined in Lemma 8 applied to the set S. Although the matrices constructed in Algorithm 1 are always part of a pair (S', B), in this proof we consider only the matrices B, and ignore the sets S' which are constructed alongside them.

We now inductively define a sequence $B_0, B_1, \ldots, B_m$, such that B_i is a matrix constructed by the algorithm in iteration i and $B_i \in \mathcal{L}_i$ corresponds to the output of the Lemma 8 algorithm applied to $S \cap \{v_1, \ldots, v_i\}$. Firstly, let $B_0 = \mathbf{0}_{d \times d}$, which is the initial condition for the algorithm in Lemma 8 and is constructed by Algorithm 1 on Line 5. Then, for the inductive step, we assume that B_{i-1} is the output of the Lemma 8 algorithm applied to $S \cap \{v_1, \ldots, v_{i-1}\}$ and we define B_i by case distinction.

Case 1: $v_i \notin S$**.** In this case, we set $B_i = B_{i-1}$, and notice that if B_{i-1} is in the set $\mathcal{L}_{i-1}$ constructed by Algorithm 1, then B_i must be in the set $\mathcal{L}_i$ since every matrix B in $\mathcal{L}_{i-1}$ is included in $\mathcal{L}_i$ on either Line 16 or Line 18. Since $S \cap \{v_1, \ldots, v_{i-1}\} = S \cap \{v_1, \ldots, v_i\}$, we have that B_i is the output of the algorithm defined in Lemma 8 applied to $S \cap \{v_1, \ldots, v_i\}$ by the inductive hypothesis.

Case 2: $v_i \in S$**.** In this case, we set B_i to be either B_{i-1} or $B_{i-1} + (1/p) v_i v_i^\intercal$, according to the result of the condition on Line 14 of Algorithm 1. Notice that, since the definition of p in Algorithm 1 is the same as the definition in Lemma 8, B_i corresponds to the result of applying an iteration of the algorithm in Lemma 8 with B_{i-i} and v_i. Therefore, by the induction hypothesis, B_i is equivalent to the output of the Lemma 8 algorithm applied to $S \cap \{v_1, \ldots, v_i\}$, which completes the inductive construction of $B_1, \ldots, B_m$.

Finally, since our defined B_m corresponds to the output of the algorithm in Lemma 8 applied to S, we can apply Lemma 8 to S and B_m which completes the proof. ◀

Succinct Planar Encoding with Minor Operations

Frank Kammer
THM, University of Applied Sciences Mittelhessen, Giessen, Germany

Johannes Meintrup
THM, University of Applied Sciences Mittelhessen, Giessen, Germany

Abstract

Let G be an unlabeled planar and simple n-vertex graph. Unlabeled graphs are graphs where the label-information is either not given or lost during the construction of data-structures. We present a succinct encoding of G that provides induced-minor operations, i.e., edge contractions and vertex deletions. Any sequence of such operations is processed in $O(n)$ time in the word-RAM model. At all times the encoding provides constant time (per element output) neighborhood access and degree queries. Optional hash tables extend the encoding with constant expected time adjacency queries and edge-deletion (thus, all minor operations are supported) such that any number of edge deletions are computed in $O(n)$ expected time. Constructing the encoding requires $O(n)$ bits and $O(n)$ time. The encoding requires $\mathcal{H}(n) + o(n)$ bits of space with $\mathcal{H}(n)$ being the entropy of encoding a planar graph with n vertices. Our data structure is based on the recent result of Holm et al. [ESA 2017] who presented a linear time contraction data structure that allows to maintain parallel edges and works for labeled graphs, but uses $\Theta(n \log n)$ bits of space. We combine the techniques used by Holm et al. with novel ideas and the succinct encoding of Blelloch and Farzan [CPM 2010] for arbitrary separable graphs. Our result partially answers the question raised by Blelloch and Farzan whether their encoding can be modified to allow modifications of the graph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases planar graph, r-division, separator, succinct encoding, graph minors

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.44

Related Version *Full Version*: `https://arxiv.org/abs/2301.10564` [18]

Funding *Johannes Meintrup*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 379157101.

1 Introduction

Graphs are used to model systems as entities and relationships between these entities. Many graphs that arise in real-world application are very large. This has given rise to an area of research with the aim of reducing the required space [1, 7, 8, 12, 14, 19]. Practical examples include large road-networks [26] or social-network graphs [10]. This has spawned research inquiries into compact representation of graphs, especially those that posses certain structural properties. The arguably most well-known such structural property is planarity. A graph is planar if it can be drawn in the plane without crossings. In this work we consider the problem of maintaining a succinct encoding of a given graph under edge contractions and vertex deletions, referred to as *induced-minor operations*. An edge contraction in a graph $G = (V, E)$ consists of removing an edge $\{u, v\} \in E$ from the graph and merging its endpoints to a new vertex x. Edge contractions are a vital technique in a multitude of algorithms, prominent examples include computing minimum cuts [20], practical treewidth computations [27] and maximum matchings [6]. At the end of the paper we extend our result from induced-minor operations to support all *minor operations*, which in addition to edge contractions and vertex deletions includes edge deletions. We work on unlabeled graphs, meaning that labels are either not given or lost when constructing our data structure.

Related Work. For encoding planar graphs without regards to providing fast access operations, Keeler et al. [21] showed an $O(n)$ bits representation. For a compression within the information theoretic lower bound refer to He et al. [13]. Due to Munro and Raman [24] there exists an encoding using $O(n)$ bits that allows constant-time queries which has subsequently been improved by Chiang et al. [5]. We build on the succinct representation due to Blelloch and Farzan, which allows encoding arbitrary separable graphs and subsequently allows constant-time queries [3], which builds on the work of Blanford et al. [2]. Recently it was shown that the encoding of Blelloch and Farzan can be constructed using $O(n)$ bits in $O(n)$ time [17]. Our work can be thought of as extending the encoding of Blelloch and Farzan to allow contraction operations. For edge contractions in planar graphs without regard to space-usage, Klein and Mozes [22] presented an algorithm that runs in $O(\log n)$ time per contraction. Their work is based on techniques by Brodal and Fagerberg [4]. For edge and vertex deletions Holm and Rotenberg showed a data structure that provides any number of such deletions in $O(n)$ time [16]. The state-of-the-art by Holm et al. [15] provides edge contractions in $O(n)$ time total. Their data structure allows constant time (per element output) neighborhood and degree queries and maintains parallel edges that occur due to merges, while also allowing to view the graph as "simple", i.e., skipping parallel edges when querying the neighborhoods. Using optional hashing techniques they provide expected constant time adjacency queries. Their data structure is based on the well-known notion of r-divisions [9, 11, 23], a technique we use as well. The data structure of Holm et al. uses $\Theta(n \log n)$ bits to store mappings and initially applies graph transformations that increase the number of vertices by a constant factor, making it not succinct, neither for unlabeled nor labeled graphs. Even assuming these steps could trivially be adapted to use $o(n)$ bits when encoding unlabeled graphs, they additionally construct a lookup table for storing small graphs such that each index encoding a graph with k vertices uses $\mathcal{H}(k) + \Theta(\mathcal{H}(k))$ bits of space, with $\mathcal{H}(k)$ the entropy of encoding planar graphs with k vertices. They then encode graphs of at least n total vertices using such indices, i.e., they use $\mathcal{H}(n) + \Theta(\mathcal{H}(n))$ bits for storing all such small graph, i.e., their data structure is not succinct.

Our result. Let G be an unlabeled planar and simple graph with n vertices. We present a succinct encoding of G that is able to process edge contractions and vertex deletions in $O(n)$ time for any number of such modifications. At all times, the data structure allows constant time (per element output) neighborhood and degree queries. Using optional hashing techniques, we provide constant expected time adjacency queries and can process any number of edge deletions in $O(n)$ expected time. This partially answers a question posed by Blelloch and Farzan if their encoding for arbitrary separable graphs can be extended to allow graph modifications [3]. Our data structure maintains the running time of the state-of-the-art solution by Holm et al. [15] for labeled planar graphs, while using significantly less space. The data structure of Holm et al. requires the input graph to be transformed by replacing each vertex with degree larger than a constant by a cycle-gadget, which increases the number of vertices by a linear factor[1]. Such a transformation is not necessary using our techniques, but our data structure only works for unlabeled graphs, and we are not able to maintain parallel edges that occur due to contractions, i.e., our graph is at all times simple. In the next section we present our main result in an intuitive fashion. Our new result is based on several previous data structures (Section 3) and on a table-lookup technique (Section 4). In Section 5, we describe and extend a succinct encoding technique due to Blelloch and

[1] Outlined in Appendix A.1 in the full version of their paper found on arXiv.

Farzan [3]. One of our challenges was to extend mappings used by Blelloch and Farzan to be semi-dynamic. Interestingly, for this we repurpose a graph data structure of Holm et al. [15] and use it to construct dynamic mappings between vertex labels. The dynamic mappings are described in Section 6, which we use in Section 7 to combine the results of all the previous sections to achieve our dynamic encoding. We end this publication by extending our encoding to provide vertex and edge deletions as well as adjacency and degree queries. All proofs can be found in the full version of this paper.

2 A succinct graph encoding for edge contractions

We now give an overview of our new data structure for succinctly encoding a simple unlabeled planar graph $G = (V, E)$ with n vertices and maintaining this encoding under edge contractions while allowing neighborhood and degree queries in constant time per element output. Vertex deletions and edge deletions are discussed in Section 7. Note that while we work with unlabeled graphs, internally we assume vertices are labeled as consecutive integers from 1 to n. The intuitive idea is to construct a set $X \subset V$ of *boundary vertices* such that $G[V \setminus X]$ (the vertex-induced graph on $V \setminus X$) contains multiple connected components $C_1, \ldots, C_k$ of "small" size at most r (which is defined later) with $k = O(n/r)$, and X of size $O(n/\sqrt{r})$. Based on this, we distinguish edges of three types: edges between vertices of X (called *boundary edges*) edges between vertices of one C_i (called *simple edges*) and edges between a vertex of X and a vertex of some C_i (called *mixed edges*). For each C_i denote with P_i the graph induced by all simple edges between vertices of C_i and all mixed edges with one endpoint in C_i. The set of all these P_i is known as an *r-division*, and each P_i is called a *piece*. Note that an r-division has some additional characteristics which are defined more precisely later, one such key characteristics that each P_i contains only $O(\sqrt{r})$ boundary vertices, and therefore is of size $O(|C_i| + \sqrt{r}) = O(r)$. Note that each boundary vertex is contained in multiple pieces. Assume that there is a data structure that is able to easily contract any number of edges in graphs of size $O(r)$ in time $O(r)$. As long as contractions are only carried out between simple edges, we would be able to easily construct a data structure that results in $O(n)$ runtime for any number of contractions by simply constructing this data structure for each piece individually. Problems occur when contracting boundary or mixed edges because contractions are no longer able to be carried out locally in a single piece as they affect vertices in multiple pieces. For example, when contracting a boundary edge $\{u, v\}$ this affects every P_i with $i \in \{1, \ldots, k\}$ that contains at least one of u and v. To provide such contractions we construct a data structure sketched in the following. Note that we distinguish between vertex merges of two vertices u, v and edge contractions between an edge $\{u, v\}$, with the latter being analogous to the first with the distinction that u must be adjacent to v.

For edges between vertices of the boundary X we construct a *boundary graph* $F = G[X]$, which at all times contains all boundary edges, including edges that occur due to contractions. A key invariant that we uphold is that the "status" of a vertex, i.e., if it is a boundary or a non-boundary vertex, never changes due to contractions. If a vertex is a boundary vertex initially, it will be handled internally as a boundary vertex even when it no longer is incident to boundary edges due to contractions. For non-boundary vertices we ensure that these vertices are never incident to boundary edges. For each $u \in X$ we maintain a mapping containing all i with $u \in V(P_i)$, i.e., the pieces that contains u. Now, when we contract a boundary edge $\{u, v\}$ we firstly merge v to u in all P_i that contain both u and v. For any $u \in V$ we denote with $N(u)$ the neighborhood of u. We process the aforementioned merge by

setting $N(u) := (N(u) \cup N(v)) \setminus \{u, v\}$ in P_i and removing v from P_i. In all P_i that contain only v, we simply relabel v to be u. In all P_i that do not contain v (but can contain u) we do not have to make modifications. Finally, $\{u, v\}$ is contracted in the boundary graph F as well. To maintain F we can use a "slow" data structure, as F is small.

To contract a mixed edge $\{u, v\}$ with $u \in X$ we know there is only one P_i that contains v, where we execute the contraction. This might result in u now being adjacent to some $x \in X \cap V(P_i)$, for which we add the respective edge $\{u, x\}$ to F. It helps to achieve our runtime goal of $O(n)$ for processing any number of edge contractions if we do not add this edge $\{u, x\}$ to all other pieces that contain both u and x. We therefore maintain a second invariant: that edges between boundary vertices are only contained in the boundary graph (for now ignoring the specifics of how this is maintained).

To handle contractions inside pieces, intuitively we build the same data structure we outlined here to one more time, splitting each piece in tiny pieces of size at most r', specified later. We can categorize all graphs of size at most r' by a lookup table, which allows us to encode every such tiny piece as an index into the lookup table (Section 4). For each graph of the lookup table we pre-compute all possible vertex merges. We then contract edges in constant time by simply retrieving (the index of) the contracted graph from the lookup table. Such a framework was previously used by Holm et al. [15] to maintain a planar graph under contractions, but with a space usage of $\Theta(n \log n)$ bits.

To output the neighborhood of a vertex we distinguish between two cases. First, to output the neighborhood of a vertex u that is not a boundary vertex, we can simply output the neighborhood in the only P_i that contains u. For a boundary vertex u we first output the neighborhood of u in F, which are all neighbors being boundary vertices, and then do the same for each piece P_i that contains neighbors of u.

To achieve a succinct data structure we build on a succinct encoding due to Blelloch and Farzan [3], outlined in Section 5. They construct an r-division for the input graph, and for each piece P_i of the r-division construct another r'-division. Each piece $P_{i,j}$ of this r'-division is categorized by a succinct index in a lookup table. They use succinct dictionaries to translate information between the two levels. These translation data structures and the lookup table can be realized with $o(n)$ bits. We use the encoding of Blelloch and Farzan, but enhance it with dynamic qualities (Section 5 and Section 6). This (partially) answers the question posed by Blelloch and Farzan [3], if modifications of the succinctly encoded graph are possible. A key notion is that most of the novel "building blocks" handle the small number of boundary vertices, so non-space efficient data structures are used.

▶ **Theorem 1.** *Let $\mathcal{H}(n)$ be the entropy of encoding a planar graph with n vertices and G an unlabeled simple n-vertex planar graph. There exists an encoding of G that provides induced-minor operations (i.e., vertex deletions and edge contractions) with these properties: The encoding requires $O(n)$ time to execute any number of induced-minor operations and provides neighborhood and degree operations in constant time (per element output). The encoding requires $\mathcal{H}(n) + o(n)$ bits can be initialized in $O(n)$ time and $O(n)$ bits.*

Using optional hashing techniques adapted from the data structure of Holm et al. [15, Lemma 5.15] we provide edge deletion and adjacency queries.

▶ **Corollary 2.** *The encoding of Theorem 1 can be extended to provide expected $O(1)$ time adjacency queries and process any number of minor operations in expected $O(n)$ time.*

3 Preliminaries

We denote by $[k] = \{1, \ldots, k\}$, with k any integer. Our model of computation is the word-RAM with a word length of $w = \Omega(\log n)$ bits. We work with simple unlabeled graphs. In the following let $G = (V, E)$. We use $G[V']$ with $V' \subseteq V$ to denote the vertex induced subgraph on V' of G. We also write $V(G)$ for the vertices V of G and $E(G)$ for E. A merge of a pair of two vertices $u, v \in V$ means replacing both vertices u, v with a vertex x with $N(x) = N(u) \cup N(v) \setminus \{u, v\}$. In our data structures we merge u and v by setting $N(u) := N(u) \cup N(v) \setminus \{u, v\}$ and removing v from V, i.e., x is either u or v. We then say that v *is merged to* u. Note that merging u to v is a different operation. Merging two adjacent vertices u, v is called *contracting the edge* $\{u, v\}$. We denote by n the number of vertices of the graph G under consideration. In this work we use n_G to denote the number of vertices of a given graph G. If the graph is clear from the context we simply write n.

Planar graph. A graph is planar exactly if it can be drawn in the plane such that no two edges cross. The family of planar graphs is closed under taking minors. Any simple planar graph G has $O(n)$ edges. We only work with simple graphs and from this point onward assume all (planar) graphs are simple. An operation that modifies a planar graph under consideration is *planarity preserving* if afterwards the graph is still planar. For brevity's sake, we assume all merges discussed in our work are planarity preserving unless stated otherwise. We denote with $\mathcal{H}(\cdot)$ the entropy to encode a planar graph, a function dependent on the number of vertices of the graph under consideration. It is known that $\mathcal{H}(n) = \Theta(n)$ [28].

We assume w.l.o.g. that all input graphs for our data structure are connected. If this is not the case, we add a dummy vertex and connect it to an arbitrary vertex in each connected component. After our encoding of Theorem 1 is constructed, we can simply ignore the dummy vertex and all its incident edges during any output of the provided operations. As we require this modification to be space-efficient, concretely that it only uses $O(n)$ bits of additional space during the construction and runs in $O(n)$ time, we provide an explicit lemma for this modification. Note that a succinct encoding of a planar graph with this additional dummy vertex requires only a constant number of additional bits, as $\mathcal{H}(n+1) = \mathcal{H}(n) + O(1)$.

▶ **Lemma 3.** *Let G be a simple planar graph. We can add a dummy vertex v_d to V and all edges $\{v_d, u\}$ to E with u being one vertex in each connected component of G using $O(n)$ bits in $O(n)$ time.*

Graph divisions. Let $G = (V, E)$ be a planar graph and r some integer. An *r-division* $\mathcal{R} = \{P_1, \ldots, P_k\}$ of G is a division of G into $k = O(n/r)$ edge-disjoint connected subgraphs called *pieces*. Each piece has $O(r)$ vertices. For each $P \in \mathcal{R}$ there exists a set of *boundary vertices* $\delta P \subset V(P)$ such that $u \in \delta P$ if and only if u is incident to an edge $\{u, v\} \in E$ with $v \notin V(P)$. For each P it holds that $|\delta P| = O(\sqrt{r})$. We denote with $\delta\mathcal{R} = \bigcup_{P \in \mathcal{R}} \delta P$ the set of boundary vertices of $\mathcal{R}$. For any r-division $\mathcal{R}$ we denote by k the number of pieces, and assume they are numbered from $[k]$ as $\mathcal{R} = P_1, \ldots, P_k$. We use a subscript numbering to distinguish between multiple r-divisions as follows: we use the same subscript numbering to refer to the number of parts of the r-division, i.e., we use k_i when talking about an r-division $\mathcal{R}_i$ for some integer i. Linear time algorithms for computing r-divisions exists [11, 23]. Note that an r-division requires each piece to be connected, and our encoding in some sense maintains a dynamic r-division. Due to modifications, pieces may become disconnected. We do not require each piece to be connected once the encoding is constructed, this abuses the definition of r-divisions without consequence.

Forbidden-vertex graph data structure. We use a so-called forbidden-vertex graph data structure that is initialized for a simple planar graph $G = (V, E)$ and any set $B \subseteq V$ of *forbidden vertices*. It allows modifications of G in the form of edge insertions and deletions, and in the form of vertex merges while maintaining two invariants: no edges between vertices of B exist and G is simple and planar. This data structure was described by Holm et al. [15] as a building block for their contraction data structure. We slightly modify their data structure and change some notation to match our use-cases. We use this data structure (among other things) for maintaining edges between so-called boundary vertices, as sketched in Section 2. We refer to this data structure as *forbidden-vertex graph data structure*.

For each vertex and edge managed by the data structure we can access and modify auxiliary data, which takes constant time per word written or read, if a reference to the vertex or edge is given. When merging two vertices u, v some edge $\{u, x\}$ might be removed and inserted again as $\{v, x\}$. We view this as the same edge, but with different endpoints. Meaning, auxiliary data of $\{u, x\}$ is now stored at $\{v, x\}$. If $\{v, x\}$ already existed before the merge, we can decide what to do with the data of the discarded parallel edge. Self-loops and forbidden edges that would occur due to a merge are output during the merge operation. All operations are only permitted if they preserve planarity. In the following we more precisely define the available modifications:

- **Merge.** Given are two vertices u, v with $u \neq v$. Merge v to u by setting $N(u) := N(u) \cup N(v) \setminus \{u, v\}$ and removing v and all incident edges from V. Returns a reference to u and reports and discards all parallel edges during the merge, and reports all non-parallel edges inserted to $N(u)$ during the merge. Edges that would occur between vertices of B are discarded.
- **Insert.** Given are two vertices u, v with $u \neq v$ and $\{u, v\} \notin E$. Insert the edge $e = \{u, v\}$ into E, unless both $u, v \in B$.
- **Delete.** Let $\{u, v\}$ be a given edge. Remove the edge $\{u, v\}$ from E.

▶ **Lemma 4** ([15]). *Let $G = (V, E)$ be a simple planar graph and $B \subseteq V$. A forbidden vertex graph data structure can be initialized for G and B in $O(n \log n)$ time. It provides constant time (per element output) neighborhood and adjacency queries and access to the label mappings. Edge insertion/deletion takes $O(\log n)$ time. Any number of free-assignment vertex merges are executed in $O(n \log^2 n)$ time. The data structure uses $O(n \log n)$ bits.*

To achieve the runtime outlined in Lemma 4 for vertex merges, each merge of two vertices $u, v \in V$ is processed by merging the vertex with the lowest degree to the vertex with the highest degree, i.e., we can not freely choose which vertex is merged. A simple mapping using standard data structures allows us to label the vertex $x \in \{u, v\}$ that remains after the merge either u or v. The details of this are found in full version of our paper. We refer to a merge of two vertices where we are able to freely decide the labeling of the remaining vertex after the merge as *free-assignment merge*. We henceforth assume that all merges of the data structure are free assignment merges. The following lemma summarizes this.

▶ **Lemma 5.** *Let $G = (V, E)$ be a simple planar graph and $B \subseteq V$ managed by the forbidden-vertex graph data structure of Lemma 4. Using $O(n)$ additional time for initialization and $O(n \log n)$ bits we are able to process any number of free-assignment merges in $O(n \log^2 n)$ time on G.*

Indexable dictionaries. We use a data structure called *indexable dictionary* (ID), initialized for a universe U of consecutive integers and a set $S \subseteq U$ and supports membership, rank and select queries. A rank query for some $x \in U$ returns $|\{y \in S : y < x\}|$. A select query for some integer i returns the value of $x \in U$ such that x is stored at rank i.

▶ **Lemma 6** ([25]). *Let $s \leq u$ be two integers. Given a set S of size s, which is a subset of a universe $U = [u]$, there is a succinct indexable dictionary (ID) on S that requires $\log \binom{u}{s} + o(s) + O(\log \log u)$ bits and supports rank/select on elements of S in constant time.*

We use IDs with $u = n$ and $s = O(n/\Lambda(n))$ for some function $\Lambda(n) = \omega(1)$, then each ID requires $o(n)$ bits. IDs can be can be constructed in $O(u)$ time using $O(u)$ bits [25].

4 Table lookup for small planar graphs

In this section we present our table lookup data structure for small graphs. Given an integer ℓ the table lists for every positive integer $\ell' \leq \ell$ every possible planar graph with at most ℓ' vertices. Such a lookup table was used by Blelloch and Farzan [3] as a building block for succinctly encoding planar and other separable graphs, which we outline Section 5. For every graph G encoded by the table, they provide adjacency queries and neighborhood iteration in constant time (per element). The table can be realized using $O(2^{\mathtt{poly}(\ell)})$ bits and time, including the data structures needed to provide the queries. To distinguish between all planar graphs with ℓ' vertices we requires $\mathcal{H}(\ell') + O(1)$ bits. This corresponds to an index into the computed table. The table contains $2^{\mathcal{H}(\ell)}$ entries. Everything mentioned so far was shown by Blelloch and Farzan. In the following we introduce additional operations and extensions of this lookup table. These modifications increase the size of the table by a negligible amount of bits while maintaining the same (asymptotical) runtime for constructing the table. We show that our modifications increase the size of each index encoding a graph with ℓ' vertices by $o(\ell')$ bits, which is negligible for our use case.

- Range filtered neighborhood iteration: Takes as input an index i into the lookup table, three vertices $u, a, b \in V$, with $G = (V, E)$ being the graph encoded at index i and $u \neq a \neq b$. Provides an iterator over all neighbors v of u with $a \leq v \leq b$.
- Batch edge deletion: Takes as input an index i into the lookup table, three vertices $u, a, b \in V$ with $G = (V, E)$ the graph encoded at index i. Returns the index j encoding the graph $G' = (V, E \setminus X)$ with $X = \{\{u, v\} : v \in N(u) \text{ and } a \leq v \leq b\}$.
- Label-preserving merge: Takes as input an index i into the lookup table and two vertices $u, v \in V$, with $G = (V, E)$ being the graph encoded at index i. Returns the index j encoding the graph G' obtained from G by setting $N(u) := N(u) \cup N(v) \setminus \{u, v\}$ and deleting v. Only allowed when preserving planarity.

All these operations can easily be pre-computed for one entry of the table in time $O(\text{poly}(\ell))$ using the same amount of bits. The sum of computations over every entry of the table is $O(2^{\text{poly}(\ell)})$. As a note on the label-preserving merge operation, vis-à-vis keeping the vertex v, but marking it deleted, consider the following example. We want to merge the vertices u and v in some graph $G = (V, E)$ encoded by the table with ℓ' vertices. If we would simply merge them, the vertex v no longer exists. In particular, the vertex set after the merge is $V' = V \setminus \{v\}$. As the vertex set for each graph encoded by the table is consecutively numbered from $[\ell']$, graphs with vertex set V' are possibly not encoded by the table. To remedy this, we simply keep the vertex v in the graph but mark it deleted. It remains to show how to handle 'mark v as deleted'. To each graph encoded by the table we add a dummy vertex called `deleted`. The table lists every possible planar graph with at most $\ell + 1$ vertices, where the vertex with the largest label is our dummy vertex `deleted`. To mark v as deleted during the label preserving merge, we set $N(\texttt{deleted}) := N(\texttt{deleted}) \cup \{v\}$ (and $N(v) := \{\texttt{deleted}\}$). We are now able to check if a vertex is deleted, by checking if it is adjacent to `deleted`. When we later encode a graph $G = (V, E)$ via an index into this lookup table, we encode it as the graph $G' = (V \cup \{\texttt{deleted}\}, E)$, i.e., with no vertices marked as deleted initially.

It remains to observe how one additional extra vertex increases the size of the table. The size of each entry encoded by the table stays asymptotically the same, and is thus of no concern to us. The number of entries in the table is $O(2^{\mathcal{H}(\ell+1)})$, and therefore each index into the lookup table encoding a graph with ℓ' vertices requires $\mathcal{H}(\ell'+1)+O(1)$ bits to be stored. As $\mathcal{H}(\ell'+1)=\Theta(\ell'+1)$ [28] it holds that $\mathcal{H}(\ell'+1)=\mathcal{H}(\ell')+O(1)$, which is negligible for our purpose. Therefore, our table uses $2^{\mathtt{poly}(\ell)}$ bits, which is asymptotically the same as the original table of Blelloch and Farzan. Each index of the table uses only a constant number of additional bits over the theoretical lower bound. Note that indices into the original unmodified table of Blelloch and Farzan also require this additional $O(1)$ bits. We introduce some further modifications that increase the additional space per index storing a graph with r' vertices by $o(r')$ bits, which is fine for our use-case.

When we later use the table lookup to contract edges, we do so by effectively replacing an unlabeled graph with a different unlabeled one. Without care, this can break internal labeling structures, e.g., a vertex in a graph encoded by the lookup table has an internal label of 5, and after replacing the graph it now has a label of 7. Section 6 and Section 7 show how additionally maintain a dynamic label mapping structure for "important" vertices, i.e., boundary vertices as described in Section 2. For this we require the graph encoded by the table to be partially labeled. Concretely this means that the labels remain correct for boundary vertices when replacing one graph with a different one. We store all possible labels for boundary vertices, of which there are $b=O(\sqrt{\ell})$ many. In detail, when encoding a single graph G with ℓ vertices, we do not store a single unlabeled representative of G in the table (i.e., a graph isomorphic to all labeled versions of G), but all graphs such that $\ell-b$ vertices are unlabeled, e.g., have an arbitrary internal label, and b vertices have all possible labelings in the range $0,\dots,\ell$. This increases the size of the table by a negligible factor, outined now. Due to the partial labeling we require, the number of bits needed to store an index into the table increases by $O(\log(\binom{\ell}{b}))=O(\log(\binom{\ell}{\sqrt{\ell}}))=o(\ell)$, and thus is negligible for our use-case. Later, when we modify a graph G_i encoded as an index i of lookup table (e.g., contract edges), we replace the index i with the index j such that the graph G_j encoded by j represents the modified graph with the additional characteristic that all boundary vertices of G_j have the same label in G_i. the labeling for the non-boundary vertices changes due to this, but what we maintain is that a non-boundary vertex remains mapped to non-boundary vertex, and that all boundary vertices maintain their same labeling, which is all that we require for our data-structure. This is expressed via a set of invariants defined in Section 6 and Section 7.

▶ **Lemma 7.** *Let ℓ be a positive integer. There exists a table that encodes all planar graphs with vertex set $\{1,\dots,\ell'\}$ for all integers $\ell'\leq\ell$ with the following properties. For every graph encoded by the table, (range filtered) neighborhood iteration, adjacency queries and label-preserving merge operations and batch edge deletion are provided in constant time (per element). The table can be constructed in $O(2^{\mathrm{poly}(\ell)})$ time using $O(2^{\mathrm{poly}(\ell)})$ bits. Every index of the table referencing a graph with ℓ' vertices requires $\mathcal{H}(\ell')+o(\ell')$ bits.*

5 Succinct encoding of planar graphs

We now describe the succinct encoding of unlabeled planar (and other separable) graphs due to Blelloch and Farzan [3]. We use their data structure as a basis for our encoding. Our result effectively extend their encoding with (induced-) minor operations. For this we need to give a technical overview of their encoding. Let $G=(V,E)$ be an unlabeled planar graph, $\mathcal{R}=\{P_1,\dots,P_k\}$ an r-division with $r=\log^4 n$, and for each P_i with $i\in[k]$,

let $\mathcal{R}_i = \{P_{i,1}, \ldots, P_{i,k}\}$ be an r'-division of P_i with $r' = \log^4 \log^4 n$. [2] The encoding assigns three integer labels to each vertex $u \in V$. A label in the entire graph (called *global label*) a label in each piece $P_i \in \mathcal{R}$ (called a *mini label*) and a label in each piece $P_{i,j} \in \mathcal{R}_i$ (called a *micro label*). We refer to G with the newly assigned labels as the *global graph*, each labeled $P_i \in \mathcal{R}$ as a *mini graph*, and each labeled $P_{i,j} \in \mathcal{R}_i$ as a *micro graph*. Note that boundary vertices of $\delta\mathcal{R}$ receive multiple mini labels, and analogous boundary vertices of $\delta\mathcal{R}_i$ receive multiple micro labels. We refer to the boundary vertices of $\delta\mathcal{R}$ with their assigned global labels as δG, and the set of boundary vertices of $\delta\mathcal{R}_i$ with their assigned mini labels in P_i as δP_i. For a given boundary vertex u identified by its global label, we refer to all occurrences u' (as a mini label) of u in a mini graph P_i as *duplicates*, and the same for boundary vertices of mini graphs P_i in regard to their occurrences in micro graphs. We refer to the set of (mini labels of) duplicate vertices in a mini graph P_i as ΔP_i, and analogous the set of (micro labels of) duplicate vertices in a micro graph $P_{i,j}$ as $\Delta P_{i,j}$. Global labels are consecutive integers assigned first to all non-boundary vertices and then to boundary vertices, i.e., all boundary vertices have larger labels than non-boundary vertices. Analogous for mini labels in mini graphs. Micro labels are assigned arbitrarily. For operations vertices are identified by their respective label. E.g., a neighborhood query of a vertex $u \in V$ takes the global label of u as an input and outputs the global labels of all $v \in N(u)$, analogous for queries in a mini or micro graph. Micro graphs are encoded as an index into a lookup table $\mathcal{T}$, listing all planar graphs of at most r' vertices. Technically this is realized by an array with one entry for each micro graph, which can be indexed by (i, j) when retrieving the entry for micro graph $P_{i,j}$. For our use case we replace the table of Blelloch and Farzan with the table described in Section 4, which provides additional operations. We now describe operations that the encoding provides, which are used by Blelloch and Farzan in their original publication, but are not defined outside of 1. All mappings are implemented using IDs (Lemma 6) over the universe $[n]$ combined with standard data structures such as lists and pointers. Let $u \in V$ be a vertex identified by its global label. For each such u, the encoding provides a mapping to access a list $\phi(u)$ that contains tuples (i, u') with i the index of a mini graph P_i that contains mini label u' of u. The lists are sorted in increasing order by i. Note that if u is a non-boundary vertex the mapping contains only a single tuple. For each such tuple (i, u') we can access a mapping $\phi_i^{-1}(u') = u$. For vertices u' in each mini graph P_i (identified by their mini label) analogous mappings $\phi_i(u')$ containing tuples (j, u'') with j the index of a micro graph $P_{i,j}$ that contains micro label u'' of u', and the analogous mappings $\phi_{i,j}^{-1}(u'') = u'$ are provided. We refer to all these mappings as *static translation mappings*.

Recall that Blelloch and Farzan assign micro labels in an arbitrary fashion. We instead assign the labels according to a coloring we define in the following. Let $P_{i,j}$ be the micro graph we want to label. We first assign labels to vertices that are neither a boundary vertex of $\delta\mathcal{R}$ nor of $\delta\mathcal{R}_i$, which we assign the color `simple`. Then we assign labels to vertices that are in the boundary $\delta\mathcal{R}$, but not in $\delta\mathcal{R}_i$, which we assign the color `global-boundary`. Then to vertices that are not in the boundary $\delta\mathcal{R}$, but in the boundary $\delta\mathcal{R}_i$, colored `mini-boundary`, and finally to vertices in both the boundary $\delta\mathcal{R}$ and in $\delta\mathcal{R}_i$, colored `double-boundary`. Consequently, for any four vertices of $P_{i,j}$ it holds that $a < b < c < d$ if a is colored `simple`, b is colored `global-boundary`, c is colored `mini-boundary` and d is colored `double-boundary`. For each mini graph we store the lowest labeled vertex of each color, which uses negligible space of $O((n/\log^4 \log^4 n) \log \log^4 \log^4 n) = o(n)$ bits overall.

[2] Blelloch and Farzan use $r' = \log n / \log \log n$ in their publication, but make it clear that there is a large degree in freedom as long as the choice is of size $o(\log n)$.

Combined with our novel way of assigning micro labels to vertices of micro graphs, the range-filtered neighborhood operation provided by our table allows us to implement the *color-filtered neighborhood* operation that outputs all neighbors x of a vertex u (in a micro graph $P_{i,j}$) such that all x are colored with $c \in \{\texttt{simple}, \texttt{global-boundary}, \texttt{mini-boundary}, \texttt{double-boundary}\}$. The operation runs in constant time (per element output). Using the label preserving merge operation of the lookup table we can easily provide such merges for every micro graph. Analogous for the vertex and edge deletion operation.

For a given unlabeled planar graph G we refer to the encoding described in this section as *basic encoding of* G. Kammer and Meintrup [17] have shown that the encoding can be constructed in $O(n)$ time using $O(n)$ bits. Our modifications have negligible impact on the runtime and space usage of the construction. This results in the following theorem.

▶ **Theorem 8.** *Let G be an unlabeled planar graph and $\mathcal{H}(n)$ the entropy of encoding a planar graph with n vertices. There exists a basic encoding of G into a global graph, mini graphs and micro graphs that uses $\mathcal{H}(n) + o(n)$ bits total. The basic encoding provides static translation mappings for the global graph, each mini graph and each micro graph. For each micro graph the encoding provides degree, adjacency, (color-filtered) neighborhood, (batch) edge/vertex deletion and label-preserving merge operations in $O(1)$ time. The basic encoding can be constructed in $O(n)$ time using $O(n)$ bits.*

6 Dynamic mapping data structures

For this section assume a planar graph G is given via the basic encoding of Theorem 8. We now describe a set of dynamic mapping structures, for which we outline the use-case in the following. We already mentioned in Section 2 that a vertex that is initially a boundary vertex (globally or/and in mini graphs) will never become a non-boundary vertex, and a non-boundary vertex will never become a boundary vertex due to any of our edge contractions. We construct dynamic variants of the static translation mappings for boundary vertices (in the global or in mini graphs). We ensure that the static translation mappings remain valid for all non-boundary vertices. Later these mappings are concretely constructed for the vertices of the initial graph (i.e., before any contractions are processed) and are maintained for all of these vertices throughout. Concretely this means that the sets for which we define mappings and data structures never change after initialization. Recall from Section 2 that when contracting an edge $\{u, v\} \in E$ we effectively forward the contraction operation to mini graphs that contain mini labels u' and v' of u and v respectively, and then forward the contraction to micro graphs in an analogous way. For our solution we require that these cascading merge operations are handled independently without interfering with each other.

Consider for example the case where we contract an edge $\{u, v\} \in E$ with u being a boundary vertex and v a non-boundary vertex. In this case we want to contract v to u (technical reasons for this are outlined in the next section). To fulfill this contraction, we forward a request to the mini graph P_i to merge the vertices u' and v', the respective mini labels of u and v in P_i. In the case that v' is a boundary vertex in P_i, but u' is a non-boundary vertex in P_i, we want to merge u' to v', which is in conflict with our desire to merge v to u in the global graph. The idea is to support free-assignment merges, whose realization is described in the next paragraph. This sort of conflict only pertains to vertices u that are part of the boundary δG (thus, a duplicate ΔP_i in P_i) and/or have a mini label u' in some P_i that is part of the boundary δP_i, i.e., we need to provide free-assignment merges for vertices of $\delta P_i \cup \Delta P_i$. We construct a dynamic mapping that allows us to decide, when merging two vertices $u', v' \in \delta P_i \cup \Delta P_i$, if the vertex that remains after the merge is

labeled u' or v'. This is realized by assigning each such vertex an *external mini label* and an *internal mini label*. Effectively we have no free choice on which internal mini label the vertex has after a merge, but we are free to assign a new external label. We construct a mapping $\texttt{internal}_i : \delta P_i \cup \Delta P_i \to \delta P_i \cup \Delta P_i$ that maps a given external label of a vertex of $\delta P_i \cup \Delta P_i$ to its internal label, and a mapping $\texttt{external}_i : \delta P_i \cup \Delta P_i \to \delta P_i \cup \Delta P_i$ that maps a given internal label of a vertex $\delta P_i \cup \Delta P_i$ to its external label. For all other vertices of P_i we ensure that the external and internal mini labels are identical, and therefore do not need to construct any mapping. This is explicitly defined in Invariant 2 later in this section.

▶ **Lemma 9.** *All mappings $\texttt{internal}_i$ and $\texttt{external}_i$ can be constructed in $O(n)$ time using $o(n)$ bits of space. They provide read/write access in $O(1)$ time.*

We also have the need for a dynamic version of the static translation mappings ϕ and ϕ_i for boundary vertices δG in the global graph and boundary vertices δP_i in mini graphs, and the mappings ϕ_i^{-1} and $\phi_{i,j}^{-1}$ for the duplicate vertices ΔP_i in mini graphs and $\Delta P_{i,j}$ micro graphs, outlined in the following paragraphs. Initially these are equal to the static mappings. We first describe the dynamic versions of the mappings ϕ_i^{-1} and $\phi_{i,j}^{-1}$, which we refer to as Φ_i^{-1} and $\Phi_{i,j}^{-1}$ respectively. Afterwards we describe the dynamic versions of the mappings ϕ and ϕ_i, referred to as Φ and Φ_i. To give an intuition for the use-case of these mappings, consider a contraction of an edge $e = \{u, v\} \in E$ with $u, v \in \delta G$. We contract this edge by first merging all u' and v' in the mini graphs P_i that contain both the duplicate u' of u and v' of v. In all P_i that contain only a duplicate of v' of v we need to know that the global label of v' is now (i.e., after the contraction) u instead of v, for which we use the described mappings. In all other mini graphs no changes need to be made.

▶ **Lemma 10.** *All mappings $\Phi_i^{-1} : \Delta P_i \to \delta G$ and $\Phi_{i,j}^{-1} : \Delta P_{i,j} \to \delta P_i$ can be constructed in $O(n)$ time using $o(n)$ bits of space. They provide read/write access in $O(1)$ time.*

We now describe the dynamic mappings Φ and Φ_i. As mentioned, we initially require all mappings $\Phi(u)$ to be equal to $\phi(u)$ for $u \in \delta G$ and analogously all mappings $\Phi_i(u')$ to be initially equal to $\phi_i(u')$ for $u' \in \delta P_i$ for all mini graphs P_i. To represent these mappings we construct a graph H that contains all boundary vertices $u \in \delta G$ and, for each mini graph P_i, a vertex p_i, with edges $\{u, p_i\}$ added to H exactly if u has a duplicate u' in P_i. Note that the existance of a duplicate u' in P_i means that u' has a non-boundary neighbor in P_i (initially). At each such edge we store the tuple (i, u'). We construct H using the forbidden vertex graph data structure of Lemma 4. The tuples stored at the incident edges of a vertex $u \in \delta G$ in H are exactly the set $\phi(u)$. Note that H is a minor of G and therefore planar. We can provide for all $u \in \delta G$: iteration over all elements $\Phi(u)$ (by iterating over $N(u)$ in H), insertion and removal of elements in $\Phi_i(u)$ (by inserting or removing edges in H), the merge of two sets $\Phi(u)$ and $\Phi(v)$ for some $v \in \delta G$ (by merging u to v or v to u in H). Some other similar operations are provided, outlined in detail in the next section where we concretely describe our edge contraction algorithm. For each mini graph P_i the analogous graph H_i is constructed, which manages the mappings Φ_i. An important note is that for each tuple $(i, u') \in \Phi(u)$ for $u \in \delta G$ the vertex u' is the external mini label of some vertex in P_i. As no external or internal micro labels are defined for micro graphs, each tuple $(j, u'') \in \Phi_i(u')$ for all mini graphs P_i contains the concrete micro label u'' in $P_{i,j}$. In the next section we describe our neighborhood operation, for which we require a special version of the mappings Φ, which we first motivate with an intuition. To output the neighborhood of a vertex $u \in \delta G$ we (intuitively) iterate over all $(i, u') \in \Phi(u)$ and, for each such (i, u'), iterate (and translate to global labels) over all neighbors of u' in P_i. To achieve a runtime of $O(|N(u)|)$ for this

operation, we require that each tuple (i, u') "contributes" at least one such neighbor. While this is true initially, due to edge contractions (and other modifications) the degree of each such u' can become 0. To remedy this, we store a special version of the mappings $\Phi(u)$ which we refer to as $\Phi^{>0}$, containing only tuples $(i, u') \in \Phi(u)$ such that the degree of u' is > 0. We construct the analogous mappings $\Phi_i^{>0}$ for all P_i. During contractions, we update the mappings $\Phi^{>0}$ and $\Phi_i^{>0}$ to uphold the aforementioned characteristic, which is formalized in Invariant 1. How this invariant is upheld, is discussed in the next section. We realize these mappings exactly as Φ and Φ_i, respectively, i.e., as graphs $H^{>0}$ and $H_i^{>0}$. Initially $H^{>0} = H$ and all $H_i^{>0} = H_i$, by the definition of boundary vertices in r-divisions (Section 3).

▶ **Invariant 1** (non-zero-degree invariant)**.** *For all $u \in \delta G$, each entry $(i, u') \in \Phi^{>0}(u)$ guarantees that u' has degree > 0 in mini graph P_i. For all $u' \in \delta P_i$ over all mini graphs P_i, each entry $(j, u'') \in \Phi_i^{>0}(u')$ guarantees that u'' has degree > 0 in micro graph $P_{i,j}$.*

▶ **Lemma 11.** *Graphs H, $H^{>0}$ and H_i, $H_i^{>0}$ can be constructed in $O(n)$ time and $o(n)$ bits.*

Using all data structures described in this section we uphold invariants below while running contractions on G – the details on this are described in the next section. For better readability we slightly abuse the definition of our `internal` (`external`) mappings of by assuming they return the identity function for $u' \notin \delta P_i \cup \Delta P_i$.

▶ **Invariant 2** (label-translation invariants)**.**

a. Global to external mini label and vice-versa:

I. *For each $u \in V \setminus \delta G$ and $(i, u') = \phi(u)$ it holds that u' is the external mini label of u in P_i and $\phi^{-1}(u') = u$.*

II. *For each $u \in \delta G$ and $(i, u') \in \Phi(u)$ (Φ is the dynamic version of ϕ) it holds that u' is the external mini label of u in P_i and $\Phi^{-1}(u') = u$.*

b. Internal to external mini label and vice-versa:

For each vertex $u' \in V(P_i)$ identified by its external mini label, it holds that $u^* =$ `internal`(u') is the internal mini label of u' and `external`$(u^*) = u'$.

c. Mini to micro label and vice-versa:

I. *For each $u' \in V(P_i) \setminus \delta P_i$ (identified by its internal mini label) and $(j, u'') = \phi_i(u')$ it holds that u'' is the micro label of u' in $P_{i,j}$ and $\phi_{i,j}^{-1} = u'$.*

II. *For each $u^* \in \delta P_i$ (identified by its internal mini label) and for each $(j, u'') \in \Phi_i(u^*)$ it holds that u'' is the micro label of u^* in $P_{i,j}$ and $\Phi_{i,j}^{-1}(u'') = u^*$.*

7 Towards a succinct dynamic encoding

For this section let $G = (V, E)$ be a graph encoded by the basic encoding of Theorem 8. Also, assume that the mappings of Lemma 9, 10 and 11 are constructed and available. In this section we describe our solution to support modifications of G. We denote by $\bar{G} = (\bar{V}, \bar{E})$ the graph G before any modifications are processed, e.g., contractions of edges. Analogously we define by $\bar{P}_i$, $\bar{P}_{i,j}$ the initial mini and micro graphs, respectively, with its initial vertices (as mini/micro labels). As sketched in Section 2 we handle contractions between so-called boundary edges with a *boundary graph* $F = G[\delta G]$ and analogously a *mini boundary graph* $F_i = P_i[\delta P_i]$ for each mini graph P_i. These graphs are realized via the forbidden vertex graph data structure (Lemma 4). For F we use as the set of forbidden vertices the empty set. For each F_i we use the duplicate vertices ΔP_i of P_i as the set of forbidden vertices. During initialization edges between forbidden vertices are removed. The forbidden-vertex graph data structure makes sure that this remains true after initialization. Let $\{u, v\} \in E$

be an edge. We say $\{u, v\}$ is *managed by* F if $\{u, v\} \in E(F)$, $\{u, v\}$ is *managed by* F_i if $\{u', v'\} \in E(F_i)$ with u' and v' the mini labels of u and v respectively, and finally we say $\{u, v\}$ is *managed by* $P_{i,j}$ if it contains the edge $\{u'', v''\}$ with u'', v'' being the micro labels of u and v, respectively. We uphold the following invariant:

▶ **Invariant 3** (edge-singleton invariant)**.**

a. *An edge $\{u, v\}$ is managed by F exactly if $u, v \in \delta G$.*

b. *An edge $\{u, v\}$ is managed by F_i exactly if $u', v' \in \delta P_i \setminus \Delta P_i$, with u', v' the respective mini labels of u and v in P_i. In this case, no other F_j $(j \neq i)$ also manages $\{u, v\}$.*

c. *All edges $\{u, v\}$ not managed by F or some F_i are managed by one micro graph $P_{i,j}$.*

Our construction of F and each F_i is the first step to achieve this invariant. We now give an intuition why this invariant is useful. Due to some edge contractions new edges $\{u, v\}$ might occur in G between boundary vertices (either global boundary vertices or vertices that are boundary vertices in a mini graph). We can not afford to add this edge to all mini and micro graphs that contain mini and micro labels of both u and v, respectively. Instead, we only add this edge to F or some F_i. Moreover, if edges $e \in E$ are managed multiple times, the runtime of the neighborhood operation can increase.

An important note is that the basic encoding of G does not adhere to the edge-singleton invariant from the get-go, i.e., edges managed by some F or F_i might be contained in one or more micro graphs initially. Using the batch edge deletion operation provided for micro graphs (Theorem 8) we can delete all edges that would initially violate our invariant. If this violates Invariant 1, we remove the respective entries from $\Phi^{>0}$ ($\Phi_i^{>0}$). This uses $O(n)$ time.

We refer to the combination of the basic encoding of G, the mappings of Lemma 9, Lemma 10 and Lemma 11, the boundary graph F and each mini boundary graph F_i as *succinct dynamic encoding of* G, summarized in the following corollary.

▶ **Corollary 12.** *The succinct dynamic encoding of G can be constructed in $O(n)$ time using $O(n)$ bits. After construction the encoding requires $\mathcal{H}(n) + o(n)$ bits. The encoding upholds the label-translation, edge-singleton and non-zero degree invariant.*

We now give an intuition how we implement the neighborhood operation for a vertex $u \in V$ identified by its global label. We first output all neighbors of u in F (which is $\emptyset$ if u is not a boundary vertex) and then, for all P_i that contain a mini label u' of u, compute all neighbors v' of u' in F_i (which is again $\emptyset$ if u' is not a boundary vertex) and output the respective global label v of v'. We then go to all micro graphs $P_{i,j}$ that contain a mini label u'' of u', compute all neighbors v'' of u'' in $P_{i,j}$ and output the respective global label v of v''. By the edge singleton invariant it is easy to see that each neighbor v of u in G is output exactly once by this algorithm. Invariant 2 provides the necessary translation operations. The missing details are discussed in the proof of the following lemma.

▶ **Lemma 13.** *For any $u \in V$ the neighborhood operation runs in time $O(|N(u)|)$.*

We now focus on our edge-contraction algorithm. For this we introduce one last invariant, which we call the *status invariant*. As sketched in Section 2 we require that for every vertex being a boundary vertex (either globally or in a mini graph) to remain a boundary vertex, and for every non-boundary vertex to remain a non-boundary vertex. For this we slightly abuse the definition of boundary vertices. By definition of r-divisions (Section 3) a boundary vertex $u \in \delta G$ has neighbors in more than one mini graph. Due to contractions (or other modifications) this might at some point no longer be true. Nonetheless, we still consider such a vertex to be a boundary vertex. We require that a boundary vertex remains a boundary vertex, and a non-boundary vertex remains a non-boundary vertex. For this we maintain the following invariant that depends on our slight abuse of the boundary vertex definition.

▶ **Invariant 4** (status invariant). *For every $u \in V$ and every $u' \in V(P_i)$ over all mini graphs P_i, it holds $u \in \delta G$ if and only if $u \in \delta \bar{G}$ as well as $u' \in \delta P_i$ if and only if $u' \in \delta \bar{P}_i$.*

We now give an overview of our edge-contraction algorithm, which we describe in three levels: vertex merges in micro graphs, vertex merges in mini graphs and edge contractions in the global graph. We guarantee the four invariants (Invariants 1, 2, 3 and 4) before and after each edge contraction. Technically, merges are executed in (mini) boundary graph(s) and micro graphs. Everything else is to maintain the mappings of Section 6. An edge $\{u, v\} \in E$ is contracted by determining all micro graphs $P_{i,j}$ that contain (micro labels of) u and v, all mini boundary graphs F that contain (mini labels of) u and v and check if F contains u and v. In all structures that contain u and v we merge v to u and update the mappings of Section 6. To guarantee the invariants we split the responsibilities among the three levels:

- **Global Graph-Responsibility.** Contractions in the global graph maintain Invariant 1 regarding $\Phi^{>0}$, Invariant 2.a., Invariant 3.a and Invariant 4 for all $u \in V$.
- **Mini Graph-Responsibility.** Vertex merges in a mini graph P_i maintain Invariant 1 regarding $\Phi_i^{>0}$, Invariant 2.b-c, Invariant 3.b, and Invariant 4 for all $u' \in V(P_i)$.
- **Micro Graph-Responsibility.** Vertex merges in a micro graph maintain Invariant 3.c.

Our contraction algorithm is built up from bottom-to-top, i.e., we first describe merges in micro graphs, then mini graphs (and mini boundary graphs) and edge contractions in G (and merges in F). To uphold the responsibilities of micro graphs we implement a variant of the forbidden-vertex graph data structure (Lemma 4) for micro graphs, summed up in the following lemma. To uphold Invariant 4 we are not allowed to merge a vertex v'' to a vertex u'' in a micro graph $P_{i,j}$ if $v'' \in \Delta P_{i,j}$ and $u'' \notin \Delta P_{i,j}$, which we formulate explicitly.

▶ **Lemma 14.** *For all micro graphs $P_{i,j}$ we can provide free assignment merges for each $P_{i,j}$ such that no edges $\{u'', v''\}$ exists that should be managed by F or F_i. If such an edge would occur due to the merge, it is not inserted to $P_{i,j}$ and instead returned. Computing any number of such merges among all micro graphs can be done in $O(n)$ total time. The operation upholds the micro graph-responsibility. Merging a vertex v'' to a vertex u'' is not allowed if $v'' \notin \Delta P_{i,j}$ and $u'' \in \Delta P_{i,j}$. All other (planar preserving) merges are allowed.*

We first note, whenever we call the merge operation of Lemma 14 for a micro graph $P_{i,j}$ in the next paragraphs, the operation returns edges $\{u'', v''\}$ that should be managed by F_i or F, but not $P_{i,j}$. We then translate $\{u'', v''\}$ to $\{u', v'\}$ with u' and v' the respective mini labels of u'' and v''. If the edge $\{u', v'\}$ should be managed by F_i, we insert it to F_i. Returned edges that should not be managed by F_i are instead returned after the merge operation in P_i is executed. This upholds Invariant 3 (restricted to micro and mini graphs). To uphold Invariant 1 (for mini graphs) we check, after any call to a merge of a vertex v'' to u'' in a micro graph $P_{i,j}$ if the degree of u'' changed from 0 to non-zero or vice-versa. If it does, we must possibly update the mapping $\Phi_i^{>0}(u')$ to either include the tuple (j, u'') or remove it, with u' the mini label of u''. Note that this is only done in the case that u' is a boundary vertex, as otherwise no mapping $\Phi_i^{>0}(u')$ exists.

Let $u', v' \in V(P_i)$ be two vertices identified by their external mini label. To provide a merge of u' and v' we distinguish between three cases: (M1) $u', v' \notin \delta P_i$, (M2) $u' \in \delta P_i$ and $v' \notin \delta P_i$ and (M3) $u', v' \in \delta P_i$. In Case M1 we determine the micro graph $P_{i,j}$ that contains micro labels u'' and v'' of u' and v', respectively, via the static mappings $\phi_i(u') = (j, u'')$ and $\phi_i(v') = (j, v'')$ as per Invariant 2. In $P_{i,j}$ we merge v'' to u'' exactly if v' should be merged to u', and otherwise merge u'' to v'' (Lemma 14). By this congruent choice of merge we uphold Invariant 2.c without having to modify any mappings. Since the merged vertex

remains a non-boundary vertex, Invariant 4 is guaranteed. This concludes all responsibilities of merges in mini graphs. All operations take constant time. Note that all merges of Case M1 are free-assignment merges.

Denote with $u^* = \texttt{internal}[u']$ and $v^* = \texttt{internal}[v']$ the internal mini labels of v' and u' respectively. For Case M2 we are forced to merge v^* to u^* to uphold Invariant 4, i.e., internally this merge is not a free-assignment merge. To execute the merge we determine the micro graph $P_{i,j}$ containing the micro label v'' of v^* via $\phi_i(v^*) = (j, v'')$. In $P_{i,j}$ we merge v'' to u'' (Lemma 14) with u'' the micro label of u^* in $P_{i,j}$.

Note that merging u'' to v'' is not allowed. We determine u'' via an operation we call *micro-label search procedure*, which searches for u'' by iterating over all $x'' \in N(v'') \cap \Delta P_{i,j}$ (the neighbors of v'' that are duplicates) and testing if $\Phi_{i,j}^{-1}(x'') = u^*$. If this is the case, we have found the duplicate $u'' := x''$ of u^*. Note that this operation can fail, as u'' and v'' are not guaranteed to be adjacent. In this *special case*, we instead iterate over all $x'' \in \Delta P_{i,j}$. A key characteristic to get a good runtime is that the special case only occurs if the edges $\{u, v\}$ exists in F, with u and v the global labels of u' and v', respectively, which allows us to upper bound the number of encountered special cases by $|E(F)| = O(n/\log^2 n)$ times.

Once the merge is executed in $P_{i,j}$ we must possibly update the mappings that translate between the internal and external mini labels. Recall that the mappings $\texttt{internal}$ and $\texttt{external}$ are only available for vertices of $\delta P_i \cup \Delta P_i$. In the case that $v' \in \delta P_i \cup \Delta P_i$ we are able to provide a free assignment merge as follows: if the request was to merge u' to v', we set $\texttt{internal}[v'] = u^*$ and $\texttt{external}[u^*] = v'$. Otherwise, no update is necessary. If $v' \notin \delta P_i \cup \Delta P_i$ we are not able to provide a free assignment merge, instead we are forced to merge v' to u'. We refer to this situation as the *M2 special case*. If the merge was called with the request to merge u' to v', and we are in this M2 special case, the operation is not allowed. In our use case this case never arises. Intuitively, constraints (e.g., Invariant 4) that force us to contract $\{u, v\}$ by merging v to u either "line up" with being able (or forced) to merge of v' to u' in P_i, with v' and u' the mini labels of v and u in P_i, respectively, or if they do not line up, we make use of the internal/external mappings.

Finally, we consider Case M3. In this case both u' and v' are boundary vertices with $u^* = \texttt{internal}[u']$ and $v^* = \texttt{internal}[v']$ being the internal mini labels of v' and u' respectively. As sketched in Section 2, our intuition for merging v' to u' is that we first merge all v'' to u'' in all micro graphs $P_{i,j}$ that contain both a duplicate u'' of u^* and v'' of v^*. Secondly, for all micro graphs $P_{i,j}$ that contain only a duplicate v'' of v^*, but not of u^*, we update the mappings $\Phi_{i,j}^{-1}(v'') := u^*$ and insert (i, v'') to $\Phi_i(u^*)$. Finally, we merge v^* to u^* in F_i. To describe the realization technically we introduce some additional notation. Denote with $Z_i^{u^* \cap v^*}$ the set of all triples (j, u'', v'') with $(j, u'') \in \Phi_i(u^*)$ and $(j, v'') \in \Phi_i(v^*)$, with $Z_i^{u^* \setminus v^*}$ the set of all tuples (j, u'') with $(j, u'') \in \Phi_i(u^*)$ such that no tuple $(j, \cdot)$ is contained in $\Phi_i(v^*)$, and with $Z_i^{u^* \oplus v^*}$ all tuples $(j, u'') \in \Phi_i(u^*)$ together with all tuples $(j', v'') \in \Phi_i(v^*)$ for which it holds that no tuple $(j', \cdot)$ exists in $\Phi_i(u^*)$. To execute a merge of v^* to u^* in P_i, first iterate over all triples $(j, u'', v'') \in Z_i^{u^* \cap v^*}$ and merge v'' to u'' in $P_{i,j}$, then iterate over all tuples $(j, v'') \in Z_i^{v^* \setminus u^*}$ and update all mappings $\Phi_{i,j}^{-1}(v'') := u^*$. Finally, set $\Phi_i(u^*) := Z_i^{u^* \oplus v^*}$ and merge v^* to u^* in F_i. In the proof we show how these sets occur (intuitively) "naturally" via merges in the graph H_i, which manages Φ_i.

Combining Cases M1, M2 and M3 we show the following lemma.

▶ **Lemma 15.** *All vertex merges in all mini graphs P_i are processed in $O(n)$ time and uphold their responsibilities. Edges that would occur due to a merge in P_i, but should be managed by F are returned. All merges excluding the M2 special case are free assignment merges.*

Contracting edges $\{u, v\}$ in G effectively works exactly as the vertex merges in mini graphs P_i with the exception that we do not need to maintain the translation between internal and external labels, and we do not need to provide free assignment merges for any case. We again distinguish between three cases: (G1) $u, v \notin \delta G$, (G2) $u \in \delta G$ and $v \notin \delta G$ and (G3) $u, v \in \delta G$. For Case G2 we employ a procedure we call *mini-label search procedure*, analogous to the micro-label search procedure for Case M2.

▶ **Lemma 16.** *After $O(n)$ initialization time, any number of edge contractions in G can be computed in $O(n)$ time and uphold the graph responsibility.*

We additionally provide constant time degree queries. Intuitively, we store the degree for boundary vertices (in G and each P_i) concretely, while for all other vertices Theorem 8 provides us with a degree query.

▶ **Lemma 17.** *After $O(n)$ initialization time the degree of any $u \in V$ can be queried in constant time.*

Using the same data structures we use for edge contractions, we can process any number of vertex deletions in $O(n)$ time. To delete a vertex u we delete all mini labels u' of u and all micro labels u'' of all u'. This mostly works analogously to the contraction algorithm.

▶ **Lemma 18.** *Any number of vertex deletions in G can be processed in $O(n)$ time and uphold the graph responsibility.*

We are now able to proof Theorem 1.

▶ **Theorem 1.** *Let $\mathcal{H}(n)$ be the entropy of encoding a planar graph with n vertices and G an unlabeled simple n-vertex planar graph. There exists an encoding of G that provides induced-minor operations (i.e., vertex deletions and edge contractions) with these properties: The encoding requires $O(n)$ time to execute any number of induced-minor operations and provides neighborhood and degree operations in constant time (per element output). The encoding requires $\mathcal{H}(n) + o(n)$ bits can be initialized in $O(n)$ time and $O(n)$ bits.*

Proof. Construct the dynamic encoding due to Corollary 12. Lemma 13 gives us the desired neighborhood operation and Lemma 17 the desired degree operation, Lemma 16 the desired contraction operation and Lemma 18 the desired vertex deletions. ◀

Using hash tables to implement the mappings Φ, $\Phi^{>0}$, Φ_i and $\Phi_i^{>0}$ we are able to provide expected constant time adjacency queries and is able to process any number of edge deletions in $O(n)$ expected time. Holm et al. used the same argument of replacing a mapping data structure with a hash table to show Lemma 5.15 in their work [15].

▶ **Corollary 2.** *The encoding of Theorem 1 can be extended to provide expected $O(1)$ time adjacency queries and process any number of minor operations in expected $O(n)$ time.*

References

1 Joyce Bacic, Saeed Mehrabi, and Michiel Smid. Shortest Beer Path Queries in Outerplanar Graphs. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, volume 212 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.62.

2 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, USA, 2003. Society for Industrial and Applied Mathematics. doi:10.5555/644108.644219.

3 Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In Amihood Amir and Laxmi Parida, editors, *Combinatorial Pattern Matching*, pages 138–150. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-13509-5_13.

4 Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representations of sparse graphs. In *In Proc. 6th International Workshop on Algorithms and Data Structures (WADS 99)*, pages 342–351. Springer-Verlag, 1999. doi:10.1007/3-540-48447-7_34.

5 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. doi:10.5555/365411.365518.

6 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

7 Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient Basic Graph Algorithms. In *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 288–301. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

8 Amr Elmasry and Frank Kammer. Space-efficient plane-sweep algorithms. In *27th International Symposium on Algorithms and Computation, ISAAC 2016*, volume 64 of *LIPIcs*, pages 30:1–30:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ISAAC.2016.30.

9 Greg N. Federickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987. doi:10.1137/0216064.

10 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. doi:10.1609/icwsm.v8i1.14532.

11 Michael T. Goodrich. Planar separators and parallel polygon triangulation. *J. Comput. Syst. Sci.*, 51(3):374–389, 1995. doi:10.1006/jcss.1995.1076.

12 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.

13 Xin He, Ming-Yang Kao, and Hsueh-I Lu. A fast general methodology for information-theoretically optimal encodings of graphs. *SIAM Journal on Computing*, 30(3):838–846, 2000. doi:10.1137/S0097539799359117.

14 Klaus Heeger, Anne-Sophie Himmel, Frank Kammer, Rolf Niedermeier, Malte Renken, and Andrej Sajenko. Multistage graph problems on a global budget. *Theoretical Computer Science*, 868:46–64, 2021. doi:10.1016/j.tcs.2021.04.002.

15 Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, Eva Rotenberg, and Piotr Sankowski. Contracting a Planar Graph Efficiently. In *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.50.

16 Jacob Holm and Eva Rotenberg. Good r-Divisions Imply Optimal Amortized Decremental Biconnectivity. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.42.

17 Frank Kammer and Johannes Meintrup. Space-Efficient Graph Coarsening with Applications to Succinct Planar Encodings. In *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ISAAC.2022.62.

18 Frank Kammer and Johannes Meintrup. Succinct planar encoding with minor operations, 2023. arXiv:2301.10564.

19 Frank Kammer, Johannes Meintrup, and Andrej Sajenko. Space-efficient vertex separators for treewidth. *Algorithmica*, 84(9):2414–2461, 2022. doi:10.1007/s00453-022-00967-3.

20 David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 21–30. Society for Industrial and Applied Mathematics, 1993. doi:10.5555/313559.313605.

21 Kenneth Keeler and Jeffery Westbrook. Short encodings of planar graphs and maps. *Discrete Applied Mathematics*, 58(3):239–252, 1995. doi:10.1016/0166-218X(93)E0150-W.

22 Philip N. Klein and Shay Mozes. *Optimization algorithms for planar graphs*. planarity.org, 2017. URL: http://planarity.org.

23 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *STOC '13: Proceedings of the forty-fifth annual ACM symposium on Theory of Computing*. Association for Computing Machinery, 2013. doi:10.1145/2488608.2488672.

24 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.

25 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.

26 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.

27 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *J. Comb. Optim.*, 37(4):1283–1311, 2019. doi:10.1007/s10878-018-0353-z.

28 György Turán. On the succinct representation of graphs. *Discret. Appl. Math.*, 8(3):289–294, 1984. doi:10.1016/0166-218X(84)90126-4.

Improved Approximation Algorithm for Capacitated Facility Location with Uniform Facility Cost

Mong-Jen Kao
Department of Computer Science, National Yang-Ming Chiao-Tung University, Hsinchu, Taiwan

Abstract

We consider the hard-capacitated facility location problem with uniform facility cost (CFL-UFC). This problem arises as an indicator variation between the general CFL problem and the uncapacitated facility location (UFL) problem, and is related to the profound capacitated k-median problem (CKM).

In this work, we present a rounding-based 4-approximation algorithm for this problem, built on a two-staged rounding scheme that incorporates a set of novel ideas and also techniques developed in the past for both facility location and capacitated covering problems. Our result improves the decades-old LP-based ratio of 5 for this problem due to Levi et al. since 2004. We believe that the techniques developed in this work are of independent interests and may further lead to insights and implications for related problems.

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

Keywords and phrases Capacitated facility location, Hard capacities, Uniform facility cost

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.45

Related Version *Full Version*: https://arxiv.org/abs/2102.06613

Funding *Mong-Jen Kao*: Supported in part by National Science and Technology Council (NSTC), Taiwan, under Grants 111-2221-E-A49-118-MY3, 112-2628-E-A49-017-MY3, and 112-2634-F-A49-001-MBK.

1 Introduction

We consider the facility location problem with hard capacities and uniform facility cost (CFL-UFC). In this problem, we are given a set $\mathcal{F}$ of facilities, a set $\mathcal{D}$ of clients, and a distance metric c defined over $\mathcal{F} \cup \mathcal{D}$. Each facility $i \in \mathcal{F}$ is associated with a uniform open cost w and a capacity u_i, which is the number of clients facility i can serve when opened up. The cost of assigning a client to a facility is equal to the distance between them. The goal of this problem is to compute a set of facilities $A \subseteq \mathcal{F}$ to open up and an assignment function $h\colon \mathcal{D} \to A$ that respects the capacity limits of the facilities in A such that, the total facility open cost plus the assignment cost, $w \cdot |A| + \sum_{j \in \mathcal{D}} c_{j,h(j)}$, is minimized.

The CFL-UFC problem originates as an important variation of the classic capacitated facility location problem (CFL), in which the open cost of each facility can be non-uniform, and is deeply related to the profound capacitated k-median problem (CKM). To better illustrate the literature of CFL-UFC, in the following we start with the introduction for the CFL problem. Then we describe the implicit connection of CFL-UFC to CKM.

The classic problem of CFL was first addressed by Shmoys, Tardos, and Aardal [15]. Since then, almost all results for this problem were based on local search heuristics, e.g., [3,13,14,18], and the best ratio known for this problem is 5, due to Bansal et al. [3].

In contrast to the rich LP-based toolsets developed for the uncapacitated facility location problem (UFL), the fact that no LP-based algorithm with constant approximation guarantee for CFL was known for a long time was intriguing. In fact, devising an LP-based approximation with $O(1)$ guarantee for CFL was listed as one of ten open problems in the textbook due to Williamson and Shmoys [17]. This problem was resolved by the notable work of An

et al. [2], in which a novel multi-commodity flow network (MFN) relaxation was presented. In a follow-up work, Kao [10] presented an iterative rounding approach and showed that, the integrality gap of the MFN relaxation is at most $(10 + \sqrt{67})/2 \approx 9.0927$.

In the pursuit of settling down the approximability of the CFL problem, an important variation between the general CFL problem and the UFL problem is when we have uniform facility cost, i.e., CFL-UFC. This was studied by Levi et al. [11], in which a 5-approximation via LP-rounding was presented. On the other hand, Aardal et al. [1] presented a 4.562-approximation based on local search technique. Interestingly, the ratio of 5 due to Levi et al. [11] also remains the best LP-based guarantee for CFL-UFC for a surprisingly long period of time since 2004.

Comparing to local search algorithms, which often has the characteristics of being simple and elegant, LP-based methods have the irreplaceable advantage of being applicable to related problems to yield new results. One of such examples is the celebrating successful story of LMP-algorithms for UFL to be combined with bi-point rounding algorithms to yield state-of-the-art results for the uncapacitated k-median problem, see, e.g., [4–8, 12, 16].

So far, the phenomenon is very different for the capacitated version of facility location and k-median problems. One primary reason is that, very little is known regarding LP-based results for both CFL and CFL-UFC.

1.1 Our Contribution

In this work, we present an LP-based 4-approximation algorithm for CFL-UFC. Our main result is the following theorem.

▶ **Theorem 1.** *There is a rounding-based algorithm for CFL-UFC that produces a 4-approximation in polynomial-time.*

Theorem 1 improves the decades-old LP-based guarantee of 5 due to Levi et al. [11] since 2004 and shows that the integrality gap of natural LP for this problem is at most 4. The algorithm we present is built on a two-staged rounding scheme that incorporates a set of novel ideas together with results and techniques developed in the past for both facility location and capacitated covering problems [9, 11].

In the following, we describe the ideas we use to obtain the 4-approximation guarantee. We believe that, the techniques developed in this work are of independent interests and may lead to further insights and progress for further related problems.

Overview of our Algorithm and Techniques

The core part of our result can be seen as a delicate orchestration of rounding procedures for the large and small instances incurred in the LP solution. In our procedure, we aim at fractionally serving the clients while making sure that the rounded facilities of interests are sufficiently sparsely-loaded so that a reasonable round-up of the assignments can be made to fully-serve the clients. Intuitively, this is possible for small facilities as they are sparsely-loaded by default. The large facilities, however, do not allow such a round-up in general since they can be tightly-loaded by assignments made in the LP solution.

To overcome this issue, we introduce the concept of *client redistribution*: When the residue demand of a client drops below a target threshold, we discard the client and redistribute part of it to the large facilities in the vicinity, defined by the LP solution, to form the so-called "*outlier clients*." The outlier clients participate in the rounding process after created and act as normal clients except for that, there is no threshold for them to be discarded, and

we guarantee that they will be fully-assigned for the final feasibility. Moreover, the way the outlier clients are created also guarantees that, the resulting assignment cost does not increase too much.

The concept of client redistribution resolves the assignment of the clients. However, when an outlier client is selected to form a cluster, we are no longer able to guarantee the overall rounding cost of the facilities, since the total facility value in that cluster can be arbitrarily small, rendering the rounding error unbounded. To prevent this undesirable situation, we introduce a matching-yielding LP technique and leave the rounding decisions for the *outlier clusters* as a global optimization problem to be resolved in the second stage of the algorithm.

In the second stage of the algorithm, we formulate the rounding problem of the remaining outlier clusters as a carefully designed assignment LP. We deploy a technique, that was originally developed for the capacitated covering problems [9], to show that, basic feasible solutions of this simple LP corresponds naturally to a matching from the non-integral facilities to the large facilities at which the outlier clients reside, and hence the rounding cost of these facilities can be bounded. Together this yields a bound for our final unconditional rounding.

Organization of this paper

This paper is organized as follows. In Section 2, we formally define CFL-UFC and describe preliminaries necessary to present our approximation algorithms. We present our approximation algorithm for CFL-UFC in Section 3 and the analysis in Section 4. Due to space limit, technical details and proofs omitted from the main content will be provided in the full version of this paper.

2 Preliminaries

In the CFL-UFC problem, we are given a set $\mathcal{F}$ of facilities, a set $\mathcal{D}$ of clients, and a distance metric c defined over $\mathcal{F} \cup \mathcal{D}$. Each $i \in \mathcal{F}$ is associated with a uniform open cost w and a capacity u_i, which is the number of clients facility i can serve when opened up. A feasible solution for CFL-UFC consists of a multiplicity function $y\colon \mathcal{F} \to \{0,1\}$ and an assignment function $x\colon \mathcal{F} \times \mathcal{D} \to \{0,1\}$ such that the following conditions are met:

- $\sum_{i\in\mathcal{F}} x_{i,j} \geq 1$, for any $j \in \mathcal{D}$, i.e., each client is assigned to some facility.
- $\sum_{j\in\mathcal{D}} x_{i,j} \leq u_i \cdot y_i$, for any $i \in \mathcal{F}$, i.e., the capacity limit of any facility is not violated.
- $x_{i,j} \leq y_i$, for any $i \in \mathcal{F}, j \in \mathcal{D}$, i.e., assignments can only be made to opened facilities.

The cost of the solution (x, y) is defined to be $\psi(x, y) := \sum_{i\in\mathcal{F}} w \cdot y_i + \sum_{i\in\mathcal{F},\, j\in\mathcal{D}} c_{i,j} \cdot x_{i,j}$. Note that, by properly rescaling the distance metric c, we may assume that $w = 1$. Given an instance $\Psi = (\mathcal{F}, \mathcal{D}, \mathbf{c}, \mathbf{u})$ of CFL-UFC, the goal of this problem is to compute a feasible solution (x, y) such that $\psi(x, y)$ is minimized.

LP relaxation and the definition of Vicinity

A natural LP relaxation for CFL-UFC and its dual LP is given below in Figure 1. It follows that, for optimal solutions (x, y) and $(\alpha, \beta, \Gamma, \eta)$ for LP-(N) and LP-(DN), $\alpha_j \geq c_{i,j}$ holds for any $i \in \mathcal{F}, j \in \mathcal{D}$ with $x_{i,j} > 0$. We will use the fact that α_j is a valid estimation on the assignment radius for any $j \in \mathcal{D}$ in x.

For the ease of presentation, in our algorithm, we use the following notion of vicinity that is defined with respect to any given assignment function x. For any $A \subseteq \mathcal{F}$ and any $j \in \mathcal{D}$, we use $N_{(A,x)}(j) := \{\, i \in A : x_{i,j} > 0 \,\}$ to denote the set of facilities in A to which j is assigned to in x. Similarly, for any $B \subseteq \mathcal{D}$ and any $i \in \mathcal{F}$, we use $N_{(B,x)}(i) := \{\, j \in B : x_{i,j} > 0 \,\}$ to denote the set of clients in B that is assigned to i in x.

LP-(N)

$$\min \sum_{i\in\mathcal{F}} y_i + \sum_{i\in\mathcal{F},j\in\mathcal{D}} c_{i,j}\cdot x_{i,j}$$

$$\sum_{i\in\mathcal{F}} x_{i,j} \geq 1, \qquad \forall j\in\mathcal{D},$$

$$\sum_{j\in\mathcal{D}} x_{i,j} \leq u_i\cdot y_i, \qquad \forall i\in\mathcal{F},$$

$$0 \leq x_{i,j} \leq y_i, \qquad \forall i\in\mathcal{F}, j\in\mathcal{D},$$

$$0 \leq y_i \leq 1, \qquad \forall i\in\mathcal{F}.$$

LP-(DN)

$$\max \sum_{j\in\mathcal{D}} \alpha_j - \sum_{i\in\mathcal{F}} \eta_i$$

$$\alpha_j \leq \beta_i + \Gamma_{i,j} + c_{i,j}, \qquad \forall i\in\mathcal{F}, j\in\mathcal{D},$$

$$u_i\cdot\beta_i + \sum_{j\in\mathcal{D}} \Gamma_{i,j} \leq 1 + \eta_i, \qquad \forall i\in\mathcal{F},$$

$$\alpha_j,\ \beta_i,\ \Gamma_{i,j},\ \eta_i \geq 0, \qquad \forall i\in\mathcal{F}, j\in\mathcal{D}.$$

Figure 1 LP relaxations for CFL-UFC.

3 4-Approximation for CFL-UFC

In the following, we describe our approximation algorithm $\mathcal{A}$ for CFL-UFC and prove Theorem 1. Let $\Psi = (\mathcal{F}, \mathcal{D}, \mathbf{c}, \mathbf{u})$ be an instance of CFL-UFC, and let $(\boldsymbol{x}', \boldsymbol{y}')$, $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\Gamma}, \boldsymbol{\eta})$ be optimal solutions for LP-(N) and its dual LP-(DN) on Ψ.

Let $I := \left\{ i \in \mathcal{F} : 0 < y'_i < \frac{1}{2} \right\}$ and $U := \left\{ i \in \mathcal{F} : y'_i \geq \frac{1}{2} \right\}$ be the sets of small and large facilities. Let $J^{(I)}$ and $J^{(\leftrightarrow)}$ be the clients that are served merely by I and the clients that are served jointly by I and U, respectively. We round up the facilities in U directly and keep the assignments made to them unchanged. What remains is the rounding problem for I and $J^{(I)} \cup J^{(\leftrightarrow)}$.

Our rounding process for I and $J^{(I)} \cup J^{(\leftrightarrow)}$ consists of two stages. In the first stage, it proceeds in iterations to select clients and form clusters. Depending on the status of the client selected, the rounding decision for the cluster may be postponed. In the second stage, our rounding process formulates the rounding decisions of the postponed clusters as a global optimization problem and makes an overall rounding decision. In the following, we describe the two stages in details.

3.1 The First Stage of the Rounding Process

In this stage, the algorithm proceeds in iterations to form clusters. Let F' and D' be the set of facilities and the set of clients remained to be processed. Initially, $F' := I$ and $D' := J^{(I)} \cup J^{(\leftrightarrow)}$. The algorithm will maintain a rounded assignment function x^* during this stage. Initially $x^* := 0$.

In each iteration, the algorithm first checks if $\sum_{i\in F'} x'_{i,j} \geq 1/2$ holds for all $j \in D'$. Intuitively, from the LP constraints, this condition guarantees that

$$\sum_{i\in N_{(F',x')}(j)} y'_i \geq \sum_{i\in F'} x'_{i,j} \geq \frac{1}{2},$$

and there will be a decent amount of facility values to be aggregated in the vicinity of client j in F' for any $j \in D'$. If not, the algorithm makes it so by repeating the following steps:

1. Pick an arbitrary $j \in D'$ with $\sum_{i\in F'} x'_{i,j} < 1/2$.
2. Apply the procedure CREATE_OUTLIER(j),
   which we later describe, to create a set of *outlier clients* for j.
3. Remove the client j from D'.

Intuitively, each client j that is picked here will belong to the set $J^{(\leftrightarrow)}$, and via replacing j with its outlier copies created by the procedure CREATE_OUTLIER(j), the rounding cost for the remaining part of j will be charged to the large facilities to which j is assigned to, i.e., $N_{(U,x')}(j)$.

The algorithm additionally maintains two sets H and H', where H denotes the set of outlier clients created in this step and $H' \subseteq H$ denotes those that have been created but not yet processed by the rounding process. Initially $H := \emptyset$ and $H' := \emptyset$.

When the condition $\sum_{i \in F'} x'_{i,j} \geq 1/2$ holds for all $j \in D'$, the algorithm applies another procedure FORM_CLUSTER, which we will later describe, to select a client from $D' \cup H'$ and form a cluster centered at that client. Depending on whether or not the selected client is outlier, the rounding decision for the cluster created may be postponed to the second stage. The procedure then removes the corresponding parts of the cluster from the residual instance $(F', H', \boldsymbol{x}', \boldsymbol{y}')$.

When the procedure FORM_CLUSTER is done, for each client $j \in D' \cap J^{(I)}$ with $\sum_{i \in F'} x'_{i,j} < 1/2$, the algorithm removes j from D' and sets $x'_{i,j}$ to be zero for all $i \in F'$. Intuitively, for each client j that is picked in this step, the algorithm guarantees that, more than half of its demand has already been assigned to a rounded facility. Hence the remaining part can be discarded.

Then the algorithm iterates to the next iteration until $D' \cup H'$ becomes empty. The following high-level pseudo-code summarizes the first stage of our rounding process.

- Repeat until $D' \cup H' = \emptyset$, do
  1. Repeat until $\sum_{i \in F'} x'_{i,j} \geq 1/2$ for all $j \in D'$, do
     - Pick an arbitrary $j \in D'$ with $\sum_{i \in F'} x'_{i,j} < 1/2$.
     - Apply CREATE_OUTLIER(j) and remove j from D'.
  2. Apply the procedure FORM_CLUSTER().
  3. For all $j \in D' \cap J^{(I)}$ with $\sum_{i \in F'} x'_{i,j} < 1/2$, remove j from D' and set $x'_{i,j}$ to be zero for all $i \in F'$.

Note that, the algorithm guarantees the invariant that, the client picked in step 1 must belong to the set $D' \cap J^{(\leftrightarrow)}$. In the following we describe the two procedures CREATE_OUTLIER and FORM_CLUSTER in details.

The procedure CREATE_OUTLIER(j).

When this procedure is called, it relocates part of the remaining demand of j to facilities in $N_{(U,x')}(j)$, i.e., the large facilities in U for which j is assigned to in $\boldsymbol{x}'$, to form outlier clients in a way as if the demand were originated from these facilities. Then it updates the assignment function $\boldsymbol{x}'$ and the dual variables $\boldsymbol{\alpha}$ accordingly.

Before describing the detail of this procedure, we describe the intuitions in the following. The outlier clients created by this procedure will be used to replace the remaining part of j, and the construction will serve for two purposes.

- First, since $\sum_{i \in F'} x'_{i,j} < 1/2$ when this procedure is called, $\sum_{i \in N_{(F',x')}(j)} y'_i$ can be less than $1/2$. Therefore, when the outlier clients of j are selected to form clusters in later iterations of the rounding algorithm, we will use the facility values in $N_{(U,x')}(j)$ to amend the short deficits of $\sum_{i \in N_{(F',x')}(j)} y'_i$ compared to $1/2$.
- Second, via the construction scheme of outlier clients, we will charge part of the assignment costs of the outlier clients to the assignment costs of j to the large facilities in $N_{(U,x')}(j)$. We note that, this is reflected in the setting of the dual values of the outlier clients.

In the following, we describe the procedure in details.

Let $r'_j := \min\{ \sum_{i\in F'} x'_{i,j} \ , \ \sum_{i\in U} x'_{i,j} \}$ be the amount of residue demand of j to be redistributed. For each $w \in N_{(U,x')}(j)$, we create a client j_w at the facility w with demand

$$d_{j_w} := \frac{r'_j}{\sum_{i\in U} x'_{i,j}} \cdot x'_{w,j} \text{ and set } x'_{i,j_w} := \frac{d_{j_w}}{\sum_{k\in F'} x'_{k,j}} \cdot x'_{i,j}$$

for each $i \in N_{(F',x')}(j)$. We add j_w to both H and H' and set $\alpha_{j_w} := \alpha_j + c_{w,j}$ to reflect the relocation of j_w from j to w. Note that, this ensures that α_{j_w} is still a valid estimation on the assignment radius of j_w.

Note that, by construction, we have

$$\sum_{w\in N_{(U,x')}(j)} d_{j_w} = r'_j \quad \text{and} \quad \sum_{k\in N_{(F',x')}(j)} x'_{k,j_w} = d_{j_w},$$

i.e., the designated residue demand r'_j of j is fully redistributed as outlier clients and each j_w is fully-assigned to facilities in F'.

After the outlier client j_w is created for each $w \in N_{(U,x')}(j)$, the procedure removes j from D' and set $x'_{i,j}$ to be zero for all $i \in F'$. It is clear that, the above updates on $\boldsymbol{x}'$ does not violate the capacity constraints of the facilities in F'.

The procedure FORM_CLUSTER().

When this procedure is called, it selects a client $j \in D' \cup H'$ with the minimum α_j to form a cluster. Depending on the set to which j belongs, the procedure proceeds differently.

- If $j \in H'$, then a cluster centered at j with satellite facilities in $N_{(F',x')}(j)$ is formed. We use $B(j) := N_{(F',x')}(j)$ to denote the set of satellite facilities at this moment. The procedure removes j from H' and $B(j)$ from F'. The rounding decision for this cluster is postponed to the second stage of the algorithm.
- If $j \in D'$, the procedure selects a facility $i \in N_{(F',x')}(j)$ with the maximum capacity u_i. Since

  $$\sum_{k\in N_{(F',x')}(j)} y'_k \ \geq \ \sum_{k\in F'} x'_{k,j} \ \geq \ \frac{1}{2}$$

  holds when this procedure is called, we will fractionally round y'_i to $1/2$ by aggregating both the facility values and the assignments from facilities in $N_{(F',x')}(j)$ to the selected facility i. This is done as follows.

  Let

  $$\delta_i := \left(\frac{1}{2} - y'_i \right) \cdot \frac{1}{\sum_{k\in N_{(F',x')}(j)\setminus\{i\}} y'_k}$$

  be the factor to relocate for each facility in $N_{(F',x')}(j) \setminus \{i\}$. For each facility $\ell \in N_{(F',x')}(j) \setminus \{i\}$, scale down y'_ℓ by $(1-\delta_i)$. For each $k \in N_{(D'\cup H',x')}(\ell)$, further scale down $x'_{\ell,k}$ by $(1-\delta_i)$ and increase $x^*_{i,k}$ by the same amount $x'_{\ell,k}$ is decreased in this step. Then the procedure increases $x^*_{i,k}$ by $x'_{i,k}$ for each $k \in D'$ and then removes i from F'. Intuitively, for each facility $\ell \in N_{(F',x')}(j) \setminus \{i\}$, we move δ_i fraction of its facility value and assignments to the facility i, and the assignment function $x^*_{i,k}$ records the rounded assignment of any $k \in D'$ to the selected facility i.

Properties Guaranteed in the 1^{st}-stage

To better illustrate how our rounding algorithm works, we summarize in the following the important properties guaranteed by the rounding procedure in the first stage. To be precise with notations, in the following, let $\boldsymbol{x}'^{(0)}$ denote the initial assignment, i.e., the initial $\boldsymbol{x}'$. For each outlier client $j \in H$, we extend the definition of $\boldsymbol{x}'^{(0)}$ to be the initial assignment of j when it was created. Let $(\boldsymbol{x}'^{(\mathrm{II})}, \boldsymbol{y}'^{(\mathrm{II})})$ denote the pair $(\boldsymbol{x}', \boldsymbol{y}')$ the algorithm has when it is about to enter the second stage, i.e., the pair $(\boldsymbol{x}', \boldsymbol{y}')$ when the first stage ends.

Let $G := \bigcup_{j \in H} B(j)$ be the set of satellite facilities whose rounding decisions are to be postponed in the second stage and $F^*_{D'}$ denote the set of facilities that are selected and rounded up to 1/2 by the procedure FORM_CLUSTER(). We have the following lemma.

▶ **Lemma 2.** *When first stage of the rounding algorithm ends, the following holds.*

- *For any* $i \in G$, $\sum_{j \in \mathcal{D} \cup H} x'^{(II)}_{i,j} \le u_i \cdot y'^{(II)}_i$.
- *For any* $j \in J^{(I)}$, $\sum_{i \in I} x^*_{i,j} + \sum_{i \in G} x'^{(II)}_{i,j} > 1/2$.
- *For any* $j \in J^{(\leftrightarrow)}$, $\sum_{i \in I} x^*_{i,j} + \sum_{i \in G} x'^{(II)}_{i,j} + \sum_{i \in U} x'^{(0)}_{i,j} > 1/2$.
- *For any outlier* $j \in H$, $\sum_{i \in F^*_{D'}} x^*_{i,j} + \sum_{i \in G} x'^{(II)}_{i,j} = \sum_{i \in I} x'^{(0)}_{i,j}$.

Intuitively, Lemma 2 says that, in order to guarantee the feasibility of the final assignment, for any $j \in J^{(I)}$, it suffices to scale up $\sum_{i \in I} x^*_{i,j} + \sum_{i \in G} x'^{(\mathrm{II})}_{i,j}$ by a factor at most two. Similar argument applies to any $j \in J^{(\leftrightarrow)}$, too. However, as scaling up $\sum_{i \in U} x'^{(0)}_{i,j}$ may not always be possible, we will instead guarantee that all the outlier clients are fully-assigned in the second stage. Note that this will provide extra amount of assignment needed to ensure the feasibility of j.

To be precise, for any $j \in \mathcal{D} \cup H$, define the scaling factor t'_j as follows. If $j \in \mathcal{D}$, i.e., j is a normal client, and $\sum_{i \in I} x^*_{i,j} + \sum_{i \in G} x'^{(\mathrm{II})}_{i,j} > 0$, then

$$t'_j := \frac{1}{\sum_{i \in I} x^*_{i,j} + \sum_{i \in G} x'^{(\mathrm{II})}_{i,j}} \cdot \left(1 - \sum_{i \in U} x'^{(0)}_{i,j} - r'_j\right),$$

where we recall that r'_j is the amount of demand of j that is redistributed as outlier clients for any $j \in J^{(\leftrightarrow)}$. In the remaining cases, we define $t'_\ell := 1$.

Intuitively, t'_j is the factor for which $\sum_{i \in I} x^*_{i,j} + \sum_{i \in G} x'^{(\mathrm{II})}_{i,j}$ should be scaled up in order for the client j to be fully-assigned. By the definition of t'_j and Lemma 2, we obtain the following corollary.

▶ **Corollary 3.** $0 \le t'_j \le 2$ *for all* $j \in \mathcal{D}$.

3.2 The Second Stage of the Rounding Process

In the second stage, the algorithm formulates the rounding decisions left for the outlier clusters, i.e., clusters centered at outlier clients in H, as a global optimization problem.

In particular, we formulate the rounding problem as a new instance of CFL-UFC with facility set $G := \bigcup_{j \in H} B(j)$ and client set U. Each $w \in U$ is associated with a demand d_w, defined as

$$d_w := \sum_{\substack{k \in H, \\ k \text{ located at } w}} \sum_{\substack{i \in B(k), \\ \ell \in \mathcal{D} \cup H}} t'_\ell \cdot x'^{(\mathrm{II})}_{i,\ell}.$$

Intuitively, in the above definition, for each large facility $w \in U$, we consider all the outlier clusters that are centered at some outlier client located at w, and collect all the assignments within these clusters to be the demand of w. As described in the previous section, we scale up these assignment accordingly by the factor t'_ℓ for each client ℓ to meet the final feasibility.

We formulate the above instance as a carefully designed assignment LP, denoted LP-(O) and listed below. Our algorithm solves LP-(O) for a *basic optimal solution* $(\boldsymbol{x}'', \boldsymbol{y}'')$.

LP-(O)

$$\min \sum_{i \in G} y_i + \sum_{i \in G,\, j \in U} c_{i,j} \cdot x_{i,j}$$

$$\sum_{i \in G} x_{i,j} = d_j, \qquad \forall j \in U,$$

$$\sum_{j \in U} x_{i,j} \le u_i \cdot y_i, \qquad \forall i \in G,$$

$$y_i \le 1, \qquad \forall i \in G,$$

$$x_{i,j},\ y_i \ge 0, \qquad \forall i \in G,\ j \in U.$$

Properties Guaranteed in the 2^{nd}-stage

First we show that the feasible region of LP-(O) is nonempty, and the basic optimal solution $(\boldsymbol{x}'', \boldsymbol{y}'')$ can be computed. For any $w \in U$, let $H(w)$ denote the set of outlier clients located at w.

For any $w \in U$ and $i \in G$ such that $i \in B(k)$ for some $k \in H(w)$, i.e., i belongs to the clusters centered at some $k \in H(w)$, consider the bundled assignment $g_{i,w}$, defined as

$$g_{i,w} := \sum_{\ell \in \mathcal{D} \cup H} t'_\ell \cdot x'^{(\mathrm{II})}_{i,\ell}.$$

The following lemma is straightforward to verify.

▶ **Lemma 4.** $\left(\boldsymbol{g},\ 2\boldsymbol{y}'^{(II)} \right)$ *is a feasible solution for LP-(O).*

One of the key properties LP-(O) provides is that, basic feasible solutions of this LP provide a matching from the set of non-extremal facilities, i.e., $i \in G$ with $0 < y''_i < 1$, to the set of facilities in U, and hence an *unconditional roundup* can be performed on $\boldsymbol{y}''$ to obtain an integral solution. The following lemma is proved by explicitly considering the rank of the coefficient matrix.

▶ **Lemma 5.**

$$|L| \le |U|, \quad \textit{where } L := \left\{ i \in G :\ 0 < y''_i < 1 \right\}.$$

3.3 Final Output

When the rounding process ends, the algorithm defines the integral multiplicity function $\boldsymbol{y}^*$ as

$$y_i^* := \begin{cases} 1, & \text{if } i \in U \text{ or } i \in F_{D'}^*, \\ \lceil y_i'' \rceil, & \text{if } i \in G, \\ 0, & \text{otherwise.} \end{cases}$$

In particular, in addition to the large facilities in U and the facilities rounded up during the first stage of the process, the algorithm also performs an *unconditional roundup* on $\boldsymbol{y}''$ for the facilities in G. Then it solves the min-cost assignment problem on $\mathcal{D}$ and $\mathcal{F}^* := \{ i \in \mathcal{F} : \ y_i^* = 1 \}$ for an optimal integral assignment $\boldsymbol{x}^\dagger$, and outputs $(\boldsymbol{x}^\dagger, \boldsymbol{y}^*)$ as the approximation solution.

4 The Analysis

Let $\mathcal{A}$ denote the rounding algorithm in Section 3. We prove the following theorem.

▶ **Theorem 6.** *Let Ψ be an instance of CFL-UFC and $(\boldsymbol{x}', \boldsymbol{y}')$ be optimal for LP-(N) on Ψ. The rounding algorithm $\mathcal{A}$ computes in polynomial time a feasible integral solution $(x^\dagger, y^*)$ for Ψ with $\psi(x^\dagger, y^*) \leq 4 \cdot \psi(x', y')$.*

The proof is outlined as follows. In Section 4.1, We define an assignment function $\boldsymbol{x}^\circ$ and shows that $(\boldsymbol{x}^\circ, y^*)$ is feasible for LP-(N) on Ψ. This shows that the feasible region of the min-cost assignment problem for $(\mathcal{D}, \mathcal{F}^*)$ is nonempty, and hence the integral assignment $\boldsymbol{x}^\dagger$ can be computed. In Section 4.2, we establish the 4-approximation guarantee for $(\boldsymbol{x}^\circ, y^*)$. This completes the proof for Theorem 6 since $\boldsymbol{x}^\dagger$ is the optimal solution for $(\mathcal{D}, \mathcal{F}^*)$.

Notations used in the analysis

In the following, we define notations and notions to describe our rounding process precisely in the analysis.

Consider the cluster-forming procedure. Let $\mathcal{C}_{D'}$ and $\mathcal{C}_{H'}$ denote the sets of clusters centered at the non-outlier clients and outlier clients, respectively. Recall that $F_{D'}^*$ denotes the set of facilities that are selected and rounded up for the clusters in $\mathcal{C}_{D'}$. Note that, $F_{D'}^* \cap G = \emptyset$, and the set of satellite facilities $B(j)$ for each $j \in H$ forms a partition of G.

For each outlier client $j \in H$, we use $w(j)$ to denote the facility in U at which j is located. We use $p(j)$ to denote the specific parent client in $J^{(\leftrightarrow)}$ from which j is created. On the contrary, for any $j \in J^{(\leftrightarrow)}$, we use $H(j)$ to denote the set of outlier clients that are created from j. For each $w \in U$, we use $H(w)$ to denote the set of outlier clients located at w.

Recall that, we use $(\boldsymbol{x}'^{(0)}, \boldsymbol{y}'^{(0)})$ to denote the initial solution the algorithm has for Ψ. For outlier clients $j \in H$ and any $i \in \mathcal{F}$, we use $\boldsymbol{x}_{i,j}'^{(0)}$ to denote the assignment made for j to i at the moment when j is created. We use $(\boldsymbol{x}'^{(\mathrm{II})}, \boldsymbol{y}'^{(\mathrm{II})})$ to denote the pair $(\boldsymbol{x}', \boldsymbol{y}')$ the algorithm maintains when it enters the second stage.

4.1 Feasibility of the Algorithm

In this section we define an intermediate assignment $\boldsymbol{x}^\circ$ and show that $(\boldsymbol{x}^\circ, \boldsymbol{y}^*)$ is a feasible solution for LP-(N) on the input instance Ψ.

Recall that, for any $w \in U$ and $i \in G$ such that $i \in B(k)$ for some $k \in H(w)$, we define the bundled assignment $g_{i,w}$ as $g_{i,w} := \sum_{\ell \in \mathcal{D} \cup H} t'_\ell \cdot x'^{(\mathrm{II})}_{i,\ell}$. Consider the basic optimal solution $(\boldsymbol{x}'', \boldsymbol{y}'')$ for LP-(O). For each $i \in G$ and $j \in \mathcal{D} \cup H$, we define the unbundled assignment h for the original clients j as

$$h_{i,j} \;:=\; \sum_{w \in U} x''_{i,w} \cdot \frac{1}{d_w} \cdot \sum_{k \in H(w),\, i' \in B(k)} t'_j \cdot x'^{(\mathrm{II})}_{i',j}.$$

Intuitively, in h we redistribute the assignment $\boldsymbol{x}''$ back for the original clients in $\mathcal{D} \cup H$ in a proportional way. It follows that for any $j \in \mathcal{D} \cup H$,

$$\begin{aligned} \sum_{i \in G} h_{i,j} \;&=\; \sum_{i \in G,\, w \in U} x''_{i,w} \cdot \frac{1}{d_w} \cdot \sum_{k \in H(w),\, i' \in B(k)} t'_j \cdot x'^{(\mathrm{II})}_{i',j} \\ &=\; \sum_{w \in U} \sum_{k \in H(w),\, i' \in B(k)} t'_j \cdot x'^{(\mathrm{II})}_{i',j} \;=\; \sum_{i \in G} t'_j \cdot x'^{(\mathrm{II})}_{i,j}, \end{aligned} \tag{1}$$

where in the second equality we apply the first constraint of LP-(O) and in the last equality we use the fact that the set of satellite facilities for each $j \in H$ forms a partition of G.

The Assignment $\boldsymbol{x}^\circ$

Provided the above, the assignment $\boldsymbol{x}^\circ$ for each $j \in \mathcal{D}$ is defined as

$$x^\circ_{i,j} \;:=\; \begin{cases} x'^{(0)}_{i,j}, & \text{if } i \in U, \\ t'_j \cdot x^*_{i,j} \;+\; \sum_{k \in H(j)} x^*_{i,k}, & \text{if } i \in F^*_{D'}, \\ h_{i,j} \;+\; \sum_{k \in H(j)} h_{i,k}, & \text{if } i \in G, \\ 0, & \text{otherwise.} \end{cases}$$

Intuitively, the assignment of each $j \in \mathcal{D}$ in $\boldsymbol{x}^\circ$ consists of its original assignments to U and the rounded assignments for clients in $\{j\} \cup H(j)$ to facilities in $F^*_{D'} \cup G$.

The following lemma is straightforward to verify.

▶ **Lemma 7.** *$(\boldsymbol{x}^\circ, \boldsymbol{y}^*)$ is feasible for LP-(N) on the input instance Ψ.*

4.2 Approximation Guarantee

In this section we establish the 4-approximation guarantee for $(\boldsymbol{x}^\circ, \boldsymbol{y}^*)$. First, we consider the cost incurred by clusters in $\mathcal{C}_{H'}$ and $\mathcal{C}_{D'}$ separately. Then we establish the overall guarantee.

Recall that, we use $p(j)$ for $j \in H$ to denote the client in $\mathcal{D}$ from which j is created. We extend the definition and define $p(k) := k$ for any $k \in \mathcal{D}$ for notational convenience.

Moreover, for any assignment $\boldsymbol{x}$ of interest, we will use $\boldsymbol{x}|_{A,B}$ to denote the assignments made in $\boldsymbol{x}$ between $A \subseteq \mathcal{F}$ and $B \subseteq \mathcal{D} \cup H$. Similarly, for any multiplicity function $\boldsymbol{y}$ of interest, we will use $\boldsymbol{y}|_A$ to denote the multiplicity of facilities in $A \subseteq \mathcal{F}$ in $\boldsymbol{y}$.

4.2.1 The clusters in $\mathcal{C}_{H'}$

The following lemma, which regards the assignment radius of the outlier clients in H, follows directly from the construction and triangle inequality.

▶ **Lemma 8.** *For any $j \in H$ and $i \in G$ such that $x'^{(II)}_{i,j} > 0$, we have $c_{i,j} \;\leq\; \alpha_j$.*

In the following, we first bound the overall assignment cost in $\boldsymbol{x}^{\circ}|_{G,\mathcal{D}}$ in terms of that in $\boldsymbol{x}''$ and $\boldsymbol{x}'^{(\mathrm{II})}\big|_{G,\mathcal{D}\cup H}$. To this end, for any client $j \in \mathcal{D} \cup H$ and any $i \in G$, we need to bound the distance between i and $p(j)$. Let $w \in U$, $k \in H(w)$, and $i' \in B(k)$ is a satellite facility of k such that $x'^{(\mathrm{II})}_{i',j} > 0$. By the triangle inequality, we have

$$c_{i,p(j)} \;\le\; c_{i,w} \;+\; c_{i',w} \;+\; c_{i',p(j)} \;\le\; c_{i,w} \;+\; \alpha_k \;+\; c_{i',p(j)}, \tag{2}$$

where in the last inequality we apply Lemma 8. Also see Figure 2 for an illustration.

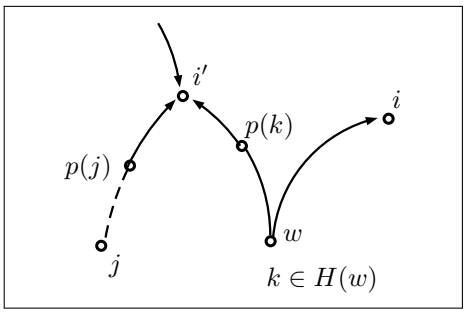


Figure 2 An illustration on the bundled assignment from $w \in U$ to $i \in G$ and unbundled assignments for $k \in H(w)$, $i' \in B(k)$ such that $x'^{(\mathrm{II})}_{i',j} > 0$.

By (2), we obtain the following lemma, which bounds the overall assignment cost in $\boldsymbol{x}^{\circ}|_{G,\mathcal{D}}$ in terms of that in $\boldsymbol{x}''$ and $\boldsymbol{x}'^{(\mathrm{II})}\big|_{G,\mathcal{D}\cup H}$.

▶ **Lemma 9.**

$$\sum_{i\in G,\, j\in\mathcal{D}} c_{i,j}\cdot x^{\circ}_{i,j} \;\le\; \sum_{i\in G,\, j\in U} c_{i,j}\cdot x''_{i,j} \;+\; \sum_{i\in G}\sum_{j\in\mathcal{D}\cup H} t'_j\cdot\big(\, c_{i,p(j)} + \alpha_j \,\big)\cdot x'^{(II)}_{i,j}.$$

In the following lemma, we expand $\boldsymbol{x}''$ and bound the overall cost incurred by $\boldsymbol{y}^*|_G$ and $\boldsymbol{x}''$ by the cost of $\boldsymbol{y}'^{(\mathrm{II})}\big|_G$, $\boldsymbol{y}'^{(0)}\big|_U$, and $\boldsymbol{x}'^{(\mathrm{II})}\big|_{G,\mathcal{D}\cup H}$.

▶ **Lemma 10.** *We have*

$$\sum_{i\in G}\lceil y''_i\rceil \;+\; \sum_{i\in G,\, j\in U} c_{i,j}\cdot x''_{i,j} \;\le\; 2\cdot\sum_{i\in G} y'^{(II)}_i \;+\; |L| \;+\sum_{i\in G}\sum_{j\in\mathcal{D}\cup H} t'_j\cdot\alpha_j\cdot x'^{(II)}_{i,j},$$

where $L := \big\{\, i\in G :\; 0 < y''_i < 1 \,\big\}$.

Applying Lemma 9, Lemma 10, Lemma 5, and the fact that $y'^{(0)}_i \ge 1/2$ for all $i \in U$, we obtain the following bound for the cost incurred by $\Big(\boldsymbol{x}^{\circ}|_{G,\mathcal{D}}\,,\,\boldsymbol{y}^*|_G\Big)$.

$$\begin{aligned}
\sum_{i\in G} y^*_i \;+\; \sum_{i\in G,\, j\in\mathcal{D}} c_{i,j}\cdot x^{\circ}_{i,j} \;&\le\; 2\cdot\sum_{i\in G} y'^{(\mathrm{II})}_i \;+\; \sum_{i\in G,\, j\in H} t'_j\cdot\big(\, c_{i,p(j)} + 2\cdot\alpha_j \,\big)\cdot x'^{(\mathrm{II})}_{i,j} \\
&\quad+\; 2\cdot\sum_{i\in U} y'^{(0)}_i \;+\; \sum_{i\in G,\, j\in\mathcal{D}} t'_j\cdot\big(\, c_{i,j} + 2\cdot\alpha_j \,\big)\cdot x'^{(\mathrm{II})}_{i,j} \\
&\le\; 2\cdot\sum_{i\in G} y'^{(\mathrm{II})}_i \;+\; \sum_{i\in G,\, j\in H}\big(\, c_{i,p(j)} + 2\cdot\alpha_j \,\big)\cdot x'^{(\mathrm{II})}_{i,j} \\
&\quad+\; 2\cdot\sum_{i\in U} y'^{(0)}_i \;+\; \sum_{i\in G,\, j\in\mathcal{D}}\big(\, 2\cdot c_{i,j} + 2\cdot t'_j\cdot\alpha_j \,\big)\cdot x'^{(\mathrm{II})}_{i,j},
\end{aligned} \tag{3}$$

where in the last inequality we apply Corollary 3 the fact that $t'_j \le 2$ for all $j \in \mathcal{D}$ and the definition that $t'_j = 1$ for all $j \in H$.

4.2.2 The clusters in $\mathcal{C}_{D'}$

Consider the cost incurred by the clusters in $\mathcal{C}_{D'}$. The following lemma, which bounds the total cost incurred by $\boldsymbol{x}^\circ|_{F^*_{D'},\mathcal{D}}$ and $\boldsymbol{y}^*|_{F^*_{D'}}$, is obtained by considering the cost of each cluster in $\mathcal{C}_{D'}$.

▶ **Lemma 11.**

(i)
$$\sum_{i\in F^*_{D'}} y^*_i \;\le\; 2\cdot\sum_{i\in I\setminus G} y'^{(0)}_i \;+\; 2\cdot\sum_{i\in G}\Big(\, y'^{(0)}_i - y'^{(II)}_i\Big). \tag{4}$$

(ii)
$$\begin{aligned}\sum_{i\in F^*_{D'},\, j\in\mathcal{D}} c_{i,j}\cdot x^\circ_{i,j} \;\le\; &\sum_{i\in F^*_{D'},\, j\in\mathcal{D}} 2\cdot t'_j\cdot\alpha_j\cdot x^*_{i,j} \;+\; \sum_{i\in F^*_{D'},\, j\in H} 2\cdot\alpha_j\cdot x^*_{i,j}\\ &+\sum_{i\in G,\, j\in\mathcal{D}} 2\cdot c_{i,j}\cdot\left(x'^{(0)}_{i,j} - \sum_{\ell\in H(j)} x'^{(0)}_{i,\ell} - x'^{(II)}_{i,j}\right) + \sum_{i\in G,\, j\in H} c_{i,p(j)}\cdot\Big(x'^{(0)}_{i,j} - x'^{(II)}_{i,j}\Big)\\ &+\sum_{i\in I\setminus G,\, j\in\mathcal{D}} 2\cdot c_{i,j}\cdot\left(x'^{(0)}_{i,j} - \sum_{\ell\in H(j)} x'^{(0)}_{i,\ell}\right) + \sum_{i\in I\setminus G,\, j\in H} c_{i,p(j)}\cdot x'^{(0)}_{i,j}. \end{aligned}\tag{5}$$

4.2.3 The overall guarantee

Combining Inequality (3), Inequality (4), and Inequality (5), and the construction scheme of outlier clients, we obtain the following lemma.

▶ **Lemma 12.**

$$\begin{aligned}\psi(\boldsymbol{x}^\circ,\boldsymbol{y}^*) \;\le\; &4\cdot\sum_{i\in U} y'^{(0)}_i \;+\; 3\cdot\sum_{i\in U,\, j\in\mathcal{D}} c_{i,j}\cdot x'^{(0)}_{i,j}\\ &+\; 2\cdot\sum_{i\in I} y'^{(0)}_i \;+\; 2\cdot\sum_{i\in I,\, j\in\mathcal{D}} c_{i,j}\cdot x'^{(0)}_{i,j} \;+\; \sum_{j\in\mathcal{D}} 2\cdot\left(1-\sum_{i\in U} x'^{(0)}_{i,j}\right)\cdot\alpha_j.\end{aligned}$$

The following lemma follows from complementary slackness between $(\boldsymbol{x}',\boldsymbol{y}')$ and $(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\Gamma},\boldsymbol{\eta})$, and the fact that $0 < y'^{(0)}_i < 1$ for all $i\in I$.

▶ **Lemma 13.**

$$\sum_{j\in\mathcal{D}}\left(1-\sum_{i\in U} x'^{(0)}_{i,j}\right)\cdot\alpha_j \;\le\; \sum_{i\in I} y'^{(0)}_i \;+\; \sum_{i\in I,\, j\in\mathcal{D}} c_{i,j}\cdot x'^{(0)}_{i,j}.$$

Applying Lemma 13 on Lemma 12, we obtain

$$\psi(\boldsymbol{x}^\circ,\boldsymbol{y}^*) \;\le\; 4\cdot\sum_{i\in\mathcal{F}} y'^{(0)}_i \;+\; 4\cdot\sum_{i\in\mathcal{F},\, j\in\mathcal{D}} c_{i,j}\cdot x'^{(0)}_{i,j},$$

and Theorem 6 is proved.

References

1 Karen Aardal, Pieter L. van den Berg, Dion Gijswijt, and Shanfei Li. Approximation algorithms for hard capacitated k-facility location problems. *Eur. J. Oper. Res.*, 242(2):358–368, 2015. doi:10.1016/j.ejor.2014.10.011.

2 Hyung-Chan An, Mohit Singh, and Ola Svensson. Lp-based algorithms for capacitated facility location. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 256–265. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.35.

3 Manisha Bansal, Naveen Garg, and Neelima Gupta. A 5-approximation for capacitated facility location. In Leah Epstein and Paolo Ferragina, editors, *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, volume 7501 of *Lecture Notes in Computer Science*, pages 133–144. Springer, 2012. doi:10.1007/978-3-642-33090-2_13.

4 Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2), March 2017. doi:10.1145/2981561.

5 Kishen N. Gowda, Thomas W. Pensyl, Aravind Srinivasan, and Khoa Trinh. Improved bi-point rounding algorithms and a golden barrier for k-median. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 987–1011. SIAM, 2023. doi:10.1137/1.9781611977554.ch38.

6 Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *J. ACM*, 50(6):795?824, November 2003. doi:10.1145/950620.950621.

7 Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 731–740, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/509907.510012.

8 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, March 2001. doi:10.1145/375827.375845.

9 Mong-Jen Kao. Iterative partial rounding for vertex cover with hard capacities. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 2638–2653, USA, 2017. Society for Industrial and Applied Mathematics.

10 Mong-Jen Kao. On the integrality gap of mfn relaxation for the capacitated facility location problem. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1071–1089, 2023. doi:10.1137/1.9781611977554.ch40.

11 Retsef Levi, David B. Shmoys, and Chaitanya Swamy. Lp-based approximation algorithms for capacitated facility location. In George L. Nemhauser and Daniel Bienstock, editors, *Integer Programming and Combinatorial Optimization, 10th International IPCO Conference, New York, NY, USA, June 7-11, 2004, Proceedings*, volume 3064 of *Lecture Notes in Computer Science*, pages 206–218. Springer, 2004. doi:10.1007/978-3-540-25960-2_16.

12 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, page 901?910, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2488608.2488723.

13 Mohammad Mahdian and Martin Pál. Universal facility location. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, volume 2832 of *Lecture Notes in Computer Science*, pages 409–421. Springer, 2003. doi:10.1007/978-3-540-39658-1_38.

14 M. Pál, É. Tardos, and T. Wexler. Facility location with nonuniform hard capacities. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, page 329, USA, 2001. IEEE Computer Society.

15 David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 265–274, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258533.258600.

16 Vincent Cohen-Addad Viallat, Fabrizio Grandoni, Euiwoong Lee, and Chris Schwiegelshohn. Breaching the 2 LMP approximation barrier for facility location with applications to k-median. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 940–986. SIAM, 2023. doi:10.1137/1.9781611977554.ch37.

17 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, USA, 1st edition, 2011.

18 Jiawei Zhang, Bo Chen, and Yinyu Ye. A multi-exchange local search algorithm for the capacitated facility location problem: (extended abstract). In George L. Nemhauser and Daniel Bienstock, editors, *Integer Programming and Combinatorial Optimization, 10th International IPCO Conference, New York, NY, USA, June 7-11, 2004, Proceedings*, volume 3064 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2004. doi:10.1007/978-3-540-25960-2_17.

The st-Planar Edge Completion Problem Is Fixed-Parameter Tractable

Liana Khazaliya
Technische Universität Wien, Austria

Philipp Kindermann
FB IV – Informatikwissenschaften, Universität Trier, Germany

Giuseppe Liotta
Department of Engineering, University of Perugia, Italy

Fabrizio Montecchiani
Department of Engineering, University of Perugia, Italy

Kirill Simonov
Hasso Plattner Institute, Universität Potsdam, Germany

Abstract

The problem of deciding whether a biconnected planar digraph $G = (V, E)$ can be augmented to become an st-planar graph by adding a set of oriented edges $E' \subseteq V \times V$ is known to be NP-complete. We show that the problem is fixed-parameter tractable when parameterized by the size of the set E'.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms

Keywords and phrases st-planar graphs, parameterized complexity, upward planarity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.46

Related Version *Full Version*: `https://arxiv.org/abs/2309.15454` [16]

Funding Research of LK supported by WWTF (Project ICT22-029); European Union's Horizon 2020 COFUND programme [LogiCS@TUWien, grant agreement No.101034440]. Research of GL and FM partially supported by MUR of Italy, under PRIN Project n. 2022ME9Z78 – NextGRAAL: Next-generation algorithms for constrained GRAph visuALization, and under PRIN Project n. 2022TS4Y3N – EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data.

Acknowledgements This research was initiated at Dagstuhl Seminar 23162: New Frontiers of Parameterized Complexity in Graph Drawing.

1 Introduction

Edge modification problems have long been a subject of investigation in graph algorithms, resulting in a vast body of literature dedicated to exploring their computational complexity (refer, for instance, to Burzyn et al. [4] and to Natanzon et al. [17] for comprehensive surveys). One specific category within this realm is the family of edge completion problems, which can be succinctly described as follows: Given a graph $G = (V, E)$ and a graph family $\mathcal{G}$, the objective is to determine whether it is possible to augment G with a set $E' \subseteq V \times V$ of edges such that $G' = (V, E \cup E') \in \mathcal{G}$. In such cases, we say that G becomes a member of $\mathcal{G}$ by *adding* the edges in E'. Edge completion problems are frequently known to be NP-hard, thereby inspiring numerous studies focusing on parameterized complexity. For a comprehensive examination of parameterized algorithms addressing edge completion problems, we point the reader to the exhaustive survey by Crespelle et al. [7].

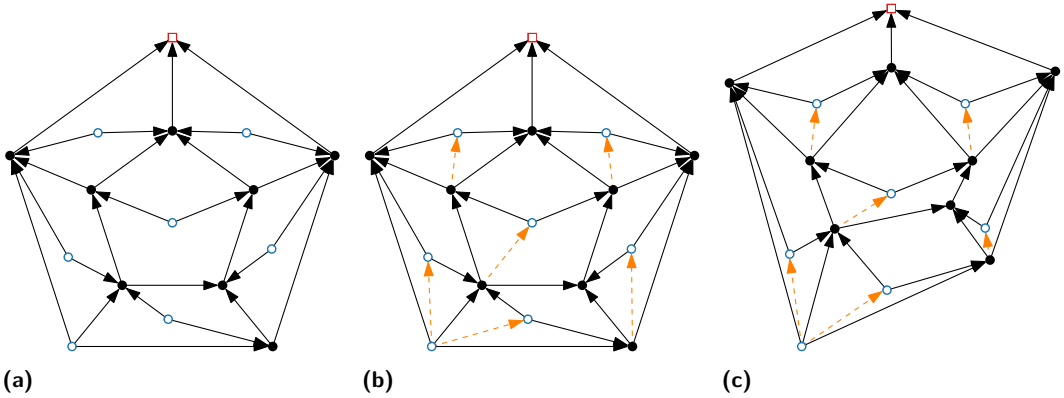


Figure 1 (a) A digraph G with $2k + 1 = 7$ sources and 1 sink; G has a unique planar embedding up to the choice of the external face; (b) A completion of G to an st-planar graph obtained by adding $2k = 6$ edges; (c) An upward planar drawing of the completion of G.

This paper focuses on the investigation of an edge completion problem specifically applied to directed graphs (*digraphs* for short). More precisely, let $G = (V, E)$ be a digraph. A vertex of G with no incoming edges is a *source* of G, while a vertex without outgoing edges is a *sink* of G. A digraph G is an *st-planar graph* if it admits a planar embedding such that: (1) it contains no directed cycle; (2) it contains a single source vertex s and a single sink vertex t; (3) s and t both belong to the external face of the planar embedding.

Upward planarity is a rather natural and well-studied notion of planarity for directed graphs (see, e.g., [5, 6, 8, 10, 13, 18]). In particular, a planar digraph is *upward* if it admits a planar drawing where all edges are oriented upward. A well-known result in graph drawing states that a digraph G is upward if and only if G is a subgraph of an st-planar graph [8, 10][1]. However, since testing for upward planarity is an NP-complete problem already for biconnected graphs [13], determining whether a biconnected graph is a subgraph of an st-planar graph is also computationally challenging. On the other hand, checking whether a digraph is st-planar can be done efficiently in polynomial time. This observation motivates for the investigation of the following problem.

st-Planar Edge Completion (st-PEC)
Input: A biconnected digraph G
Parameter: $k \in \mathbb{N}$
Question: Is it possible to add at most k edges to G such that the resulting graph is an st-planar graph?

In this paper, we present a fixed-parameter tractable algorithm for the st-Planar Edge Completion problem. To help understanding the combinatorial and algorithmic challenges behind the problem, we make the observation that the parameter k provides an upper bound on the number of sources and sinks in the input digraph G. Since an edge can remove the presence of at most one source and one sink, if the total number of sources and sinks in G exceeds $2k + 2$, we can promptly reject the instance. Conversely, a positive answer to st-Planar Edge Completion implies that G is upward planar. In this respect, it is worth

[1] From the proof in Lemma 4.1 of [10], one can in fact observe that a digraph is upward planar if and only if it is a subgraph of an st-planar graph defined over the same set of vertices.

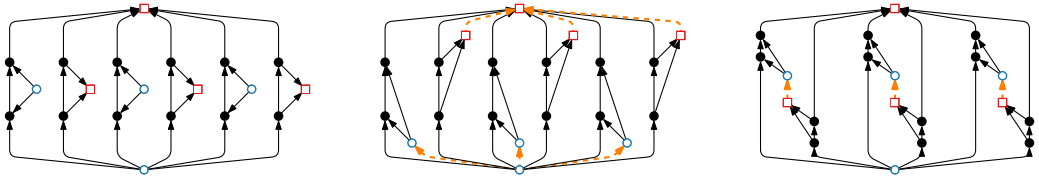


Figure 2 (a) A biconnected digraph G with 4 sources and 4 sinks; (b) With the given embedding, 6 edges have to be added to complete G to an st-planar graph; (c) With a different embedding, adding 3 edges is sufficient.

mentioning that Chaplick et al. [5] have previously demonstrated that testing a digraph for upward planarity is fixed-parameter tractable when parameterized by the number of its sources. However, for every $k \geq 1$, there are upward planar digraphs with at most $2k + 1$ sources that cannot be augmented to an st-planar graph by adding k edges; refer to Figure 1 for an illustration. Furthermore, while an upward planarity test halts upon finding an upward planar embedding, not all upward planar embeddings of the same digraph can lead to an st-planar graph after the addition of k edges. Figure 2 demonstrates an upward planar digraph along with two of its upward planar embeddings: the embedding in Figure 2a requires 6 edges to be augmented into an st-planar digraph, whereas the embedding in Figure 2c can be augmented with 3 edges.

In order to overcome the above technical challenges, our result is based on a structural decomposition of the digraph into its triconnected components using SPQR-trees (similarly as done in [5]), as well as on novel insights regarding the combinatorial properties of upward planar digraphs. Since the proof is rather technical, after giving preliminaries and basic notation in Section 2, we present an overview of the approach in Section 3. Next, the FPT algorithm is described in full detail in Section 4. We conclude in Section 5. For space reasons, the proof of statements marked with a "$\star$" are omitted and can be found in the full version of the paper [16].

2 Preliminaries

In this section, we provide basic definitions and tools that will be used throughout the paper.

Planar drawings and embeddings. A *planar drawing* of a graph G maps the vertices of G to points of the plane and the edges of G to Jordan arcs such that no two arcs share a point except at common end-vertices. A planar drawing partitions the plane into topologically connected regions called *faces*, one of which is unbounded and called the *external face*, in contrast with all other faces which are *inner faces*. For a digraph G, a planar drawing is called *upward* if each edge oriented from a vertex u to a vertex v is represented by a Jordan arc monotonically increasing from the point representing u to the point representing v. A graph (digraph) is *planar* (*upward planar*) if it admits a planar drawing (upward planar drawing). A *planar embedding* (*upward planar embedding*) $\mathcal{E}(G)$ of a planar graph (upward planar digraph) G represents an equivalence class of planar drawings (upward planar drawings) with the same inner faces and the same external face, up to a homeomorphism of the plane. Graph G is *plane* if it comes with a fixed planar embedding $\mathcal{E}(G)$.

SPQR-trees. We recall the definition of *SPQR-tree*, introduced in [8], which represents the decomposition of a biconnected graph G into its triconnected components [15]. Each triconnected component corresponds to a non-leaf node ν of T; the triconnected component itself is called the *skeleton* of ν and is denoted as $\text{skel}(\nu)$. Node ν can be: (i) an *R-node*, if $\text{skel}(\nu)$ is a triconnected graph; (ii) an *S-node*, if $\text{skel}(\nu)$ is a simple cycle of length at least three; (iii) a *P-node*, if $\text{skel}(\nu)$ is a bundle of at least three parallel edges. A degree-1 node of T is a *Q-node* and represents a single edge of G. A *real edge* (resp. *virtual edge*) in $\text{skel}(\nu)$ corresponds to a Q-node (resp., to an S-, P-, or R-node) adjacent to ν in T. Neither two S- nor two P-nodes are adjacent in T. The SPQR-tree of a biconnected graph can be computed in linear time [8, 14]. Let e be a designated edge of G, called the *reference edge* of G, let ρ be the Q-node of T corresponding to e, and let T be rooted at ρ. For any P-, S-, or R-node ν of T, $\text{skel}(\nu)$ has a virtual edge, called *reference edge* of ν and denoted as e_ν, associated with a virtual edge in the skeleton of its parent. The end-vertices of the reference edge of ν are called the *poles* of ν. For every node $\nu \neq \rho$, the *pertinent graph* G_ν *of* ν is the subgraph of G whose edges correspond to the Q-nodes in the subtree of T rooted at ν. Without loss of generality, we shall consider SQPR-trees where every S-node has exactly two children (see, e.g., [5, 9, 12]); this lifts the condition that two S-nodes cannot be adjacent in T.

Angles in upward drawings. Let $G = (V, E)$ be a digraph. For each edge $(u, v) \in E$, we write uv if (u, v) is oriented from u to v in G, and we write vu otherwise. A vertex v is a *switch* of G, if it is either a source or a sink, and it is a *non-switch* otherwise. Recall that a digraph is upward planar if and only if it is a subgraph of an st-planar graph [8]. Hence, being upward planar is a necessary condition for YES-instances of st-Planar Edge Completion. Consider now a biconnected plane digraph G. An *angle* is an incidence between a vertex v and a face f of G. Let α be one such angle, and consider the two edges incident to v that belong to the boundary of f. If such edges are one incoming and one outgoing, α is a *non-switch angle*, while if the edges are both incoming or both outgoing, α is a *switch angle*. Note that a switch angle in a face f can be made by two edges that are incident to a non-switch vertex v: it is enough that the edges of f incident to v are both incoming or both outgoing. In this case, v is a *local switch* for face f. An *angle assignment* is a labeling λ of the angles of G with labels $\{-1, 0, +1\}$ (see, e.g., [1, 2, 3, 11]). In particular, non-switch angles can only receive the label 0, while switch angles can be labeled as either -1 or $+1$. The planar embedding of G can be realized as an upward drawing if and only if there is an angle assignment such that: (i) each switch vertex has exactly one angle labeled $+1$; (ii) each non-switch vertex has exactly two angles labeled as 0, while all the others are switch angles labeled -1; (iii) the difference between the number of angles labeled $+1$ and the number of angles labeled -1 along the boundary of each inner face is -2; (iv) the difference between the number of angles labeled $+1$ and the number of angles labeled -1 along the boundary of the external face is $+2$. Observe that property (ii) implies that each non-switch vertex forms exactly two non-switch angles. An angle assignment satisfying the above properties is called *upward*. The restriction of an upward angle assignment to the angles of a single face f is an *upward angle assignment for* f.

3 Overview of the Approach

Let G be a biconnected digraph. Since testing for planarity can be done in linear time, we shall assume that G is planar. We begin by explaining two key ingredients for our algorithm, namely, the use of SPQR-trees to encode all the planar embeddings of G, and the use of

upward angle assignments to incrementally saturate G. The main crux of our algorithm lies in blending these two ingredients together to design a dynamic program that solves the problem in FPT time.

Let T be a rooted SPQR-tree of a planar graph G with reference edge e. The planar embeddings of G in which the edge e lies on the boundary of the external face can be obtained as follows (see, e.g., [8]). For a P- or R-node ν, denote by $\text{skel}^-(\nu)$ the skeleton of ν without its reference edge. If ν is a P-node, the embeddings of $\text{skel}(\nu)$ are the different permutations of the edges of $\text{skel}^-(\nu)$. If ν is an R-node, $\text{skel}(\nu)$ has two possible embeddings, obtained by flipping $\text{skel}^-(\nu)$ at its poles. No operations are needed at S- and Q-nodes.

Consider now an upward planar drawing Γ of G and hence assume that G is plane. Let λ be the upward angle assignment *induced* by Γ. Precisely, the switch angles that are larger (smaller) than π in Γ are labeled as $+1$ (-1), while the non-switch angles are labeled as 0. Let v be a source (sink) of G and let f be the face of G in which v makes its $+1$ angle. Let u be a vertex of f different from v. We say that adding uv (vu) to G *saturates* v, and that uv (vu) is a *saturating edge*. Namely, v becomes a non-switch vertex in $G' = (V, E \cup \{uv\})$. Notably, f is the only face in which an edge saturating v can be added: one easily verifies that choosing any other face would lead to a non-upward angle assignment.

Based on the previous reasoning, at high-level, the algorithm will exploit a bottom-up traversal of the SPQR-tree T to explore the planar embeddings of G. For each visited node, it will keep track of the information related to the minimum number of edges required to saturate all switches that lie in the inner faces of the corresponding pertinent graph. The interface of a candidate solution is encoded in terms of "signatures" which, informally, are strings containing all switches along the boundary of the external face of the pertinent graph that do not yet have any angle labeled as $+1$ and all vertices that must instead contribute with a -1 angle along the boundary. A running time bounded by a function of the budget k is obtained by several crucial insights about how a bounded number of switches in the graph affects the possible signatures and limits the relevant embeddings to be considered.

4 An FPT Algorithm for *st*-Planar Edge Completion

In this section, we describe our FPT algorithm, which leads to the following theorem.

▶ **Theorem 1.** *Let G be an n-vertex biconnected plane digraph. There is an algorithm that solves st-Planar Edge Completion in $2^{O(k^2)} \cdot n^2$ time.*

We begin by describing the records used by our dynamic program (Section 4.1), which are used to encode the angles along the boundary of the external face of a pertinent graph. Next, we describe the algorithm (Section 4.2), which constructs such records while traversing bottom-up the SPQR-tree of the input graph.

4.1 Setting up the Records for Dynamic Programming

Signatures. We begin with some notation and definitions. Let G be a plane digraph. Let Π_{uv} be a simple undirected path of G from a vertex u to a vertex v. The *signature* of Π_{uv} is a string Σ_{uv} computed as follows. Consider a walk along Π_{uv} from u to v. For each encountered vertex w distinct from u and v, look at the two edges incident to w in Π_{uv}. If the two edges are one incoming and one outgoing, we do not append any symbol to Σ_{uv}. If the two edges are both outgoing (incoming) and w is a switch of G, we append the symbol σ (τ). If the two edges are both outgoing (incoming) and w is not a switch of G – hence, it is a local switch for some face f –, we append the symbol σ_ℓ (τ_ℓ). Observe that, if Π_{uv} is a single

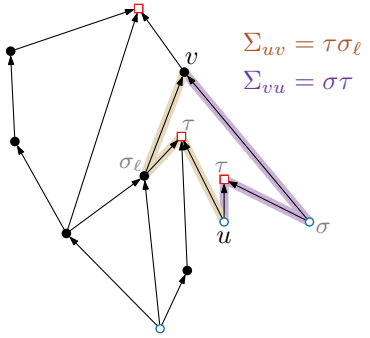


Figure 3 The signatures of two paths Π_{uv} (brown background) and Π_{vu} (purple background).

edge connecting u to v, then $\Sigma_{uv} = \emptyset$. At high level, the idea is that when walking along a piece of the boundary of some face f of G, non-switch angles are ignored as their only possible value in an angle assignment is 0. On the other hand, the symbols σ_ℓ and τ_ℓ will encode switch-angles whose only possible value is -1 (else the corresponding vertex would be a switch of G). Finally, the symbols σ and τ will point to switch angles that may be assigned either -1 or $+1$. Refer to Figure 3 for an illustration.

A signature is *short* if it contains at most $4k + 2$ symbols. Let Σ^* be the set of short signatures; we observe the following.

▶ **Observation 2.** *The cardinality of* Σ^* *is* $2^{O(k)}$.

Half-boundaries. Let G be a biconnected planar digraph, and let T be the SPQR-tree of G rooted at a Q-node representing an arbitrary edge e of G. For each node ν of T, we recall that G_ν is the pertinent graph, and we denote by u, v the poles of ν (omitting the dependency on ν for simplicity). Assuming that G_ν comes with a fixed planar embedding, let f be the external face of G_ν. The *half-boundary* B_{uv} of ν is the path containing the vertices of f encountered in a clockwise walk of the face from u to v. The half-boundary B_{vu} of ν is defined analogously walking from v to u. A vertex w on the boundary of f is *bifacial* if it belongs to both B_{uv} and B_{vu} (which implies that w is a cutvertex of G_ν and hence ν is an S-node). For each of the two half-boundaries we can define the two corresponding signatures Σ_{uv} and Σ_{vu}. We will assume that for each symbol of Σ_{uv} and Σ_{vu} we have a pointer to the corresponding vertex. Let B be one of the two half-boundaries of ν and let Σ be its signature. Let B' be a path contained in B (possibly $B = B'$). The *restriction* of Σ to B', denoted as $\Sigma[B']$, is the substring of Σ containing the symbols whose corresponding vertices belong to B'. The next lemma shows that working with short signatures is not restrictive.

▶ **Lemma 3.** *Let G be a biconnected upward planar digraph with a fixed upward planar embedding $\mathcal{E}(G)$. Let T be the SPQR-tree of G rooted at a Q-node representing an arbitrary edge e of G. Let ν be a node of T. For any fixed k, if G can be augmented to an st-planar graph by adding at most k saturating edges, then the signatures Σ_{uv} and Σ_{vu} of the two half-boundaries B_{uv} and B_{vu} of ν are both short.*

Proof. Let Γ be an upward planar drawing of G whose corresponding upward planar embedding is $\mathcal{E}(G)$, and consider the subdrawing Γ' induced by G_ν. Let λ be the upward angle assignment induced by Γ', and let f be the external face of G_ν. We know that f contains at most $2k + 2$ switches, otherwise k saturating edges would not suffice to turn G into an st-planar graph. Hence, λ can label $+1$ at most $2k + 2$ angles along the boundary of f. Also, since λ obeys to property (iv) of an upward angle assignment, it labels -1 at most $2k$ angles. Therefore, Σ_{uv} and Σ_{vu} can each contain at most $4k + 2$ symbols. ◀

Internal assignments. An angle of G_ν is *internal* if it is defined in an inner face of G_ν. An *internal assignment* of G_ν is an angle assignment λ that labels all the internal angles of G_ν and that respects properties (i)–(iii) for upward angle assignments (but ignoring property (iv)). A switch vertex of G is called *active* with respect to λ if none if its internal angles (if any) received value $+1$. The *cost* of an internal assignment λ of G_ν is the minimum number of saturating edges needed to saturate all switches of G_ν that are not active with respect to λ.

Partial solutions. We are now ready to define the table used by our dynamic program. A tuple $\langle \Sigma_1, \Sigma_2, b_1, b_2 \rangle$, such that Σ_1, Σ_2 is a pair of short signatures and b_1, b_2 is a pair of flags, is called a *candidate tuple* in the following. Given a node ν and a candidate tuple $\langle \Sigma_1, \Sigma_2, b_1, b_2 \rangle$, the function $X(\nu, \Sigma_1, \Sigma_2, b_u, b_v)$ returns the minimum cost of an internal assignment λ of G_ν such that: (1) Σ_1 and Σ_2 are the signatures of its two half-boundaries B_{uv} and B_{vu}, respectively, (2) the flag b_u is true if and only if u is an active switch with respect to λ, (3) the flag b_v is true if and only if v is an active switch with respect to λ. The set of *partial solutions* for ν is given by the restriction of X to the single node ν. Also, a pair of signatures is *empty* if both its signatures are empty (i.e., they do not contain any symbol).

4.2 Description of the Algorithm

The function X is computed by traversing T bottom-up. For each node ν of T, we initialize $X(\nu, \Sigma_1, \Sigma_2, b_1, b_2) = +\infty$ for each candidate tuple $\langle \Sigma_1, \Sigma_2, b_1, b_2 \rangle$. We only ensure that $X(\nu, \Sigma_1, \Sigma_2, b_1, b_2)$ is computed precisely if the value is at most k; for any value larger than k we assume that $X(\nu, \Sigma_1, \Sigma_2, b_1, b_2) = +\infty$ is the correct setting, since we are only interested in the solutions that add at most k edges.

If ν is a leaf node, then it is a Q-node and G_ν is a single edge. In this case, either u is the source and v is the sink of G_ν, or vice-versa. Then we set $X(\nu, \emptyset, \emptyset, \text{true}, \text{true}) = 0$.

The lemma below deals with the case in which ν is an S-node. Since S-nodes have exactly two children and are not used to describe the planar embeddings of G, the routine of the algorithm at S-nodes is relatively simple. Next, we will consider P-nodes and R-nodes, which require more involved arguments.

▶ **Lemma 4.** *Let ν be an S-node of T. The set of partial solutions of ν can computed in $2^{O(k)}$ time.*

Proof. Let μ_1 and μ_2 be the two children of ν. In order to compute the partial solutions for ν, we check whether pairs of internal assignments of G_{μ_1} and G_{μ_2} can be combined together. Let $\langle \Sigma_{1,1}, \Sigma_{1,2}, b_{1,1}, b_{1,2} \rangle$ and $\langle \Sigma_{2,1}, \Sigma_{2,2}, b_{2,1}, b_{2,2} \rangle$ be a pair of candidate tuples. Also, let $C = X(\mu_1, \Sigma_{1,1}, \Sigma_{1,2}, b_{1,1}, b_{1,2}) + X(\mu_2, \Sigma_{2,1}, \Sigma_{2,2}, b_{2,1}, b_{2,2})$.

We first verify that $C \leq k$, and that $b_{1,2} \vee b_{2,1} = \text{true}$. The first condition guarantees that we have not exceeded our budget k of saturating edges, while the second condition guarantees that the pole shared by μ_1 and μ_2 does not receive the value $+1$ twice in the final internal assignment of G_ν. If both conditions are satisfied, then we proceed as detailed below, otherwise, we discard the pair of candidate tuples.

Denote by u and w the poles of μ_1, and by w and v the poles of μ_2. Observe that B_{uv} corresponds to the union of B_{uw} and B_{wv} (vertex w is hence bifacial). Based on this observation, we show how to compute Σ_1 for B_{uv}, the computation of Σ_2 can be performed analogously. We initially set $\Sigma_1 = \Sigma_{1,1}$. Consider the two edges incident to w along B_{uv}. If one edge is incoming and the other is outgoing, then we do not append any symbol. If both edges are incoming or both outgoing, we check whether one of $b_{1,2}$ and $b_{2,1}$ is false. If so, we append the symbol σ_ℓ if w is a source of G, or the symbol τ_ℓ otherwise. If none of $b_{1,2}$ and

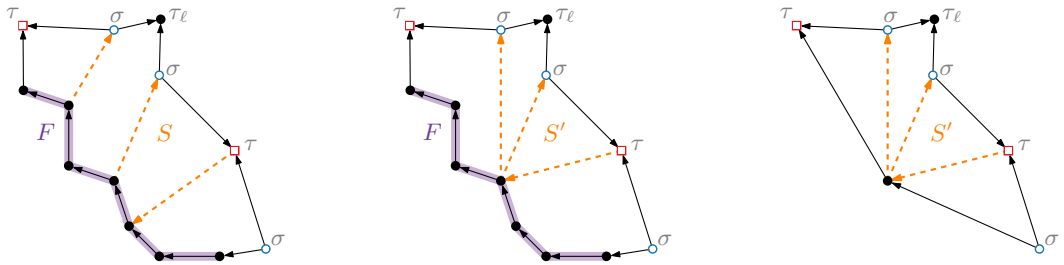


Figure 4 Illustration for Lemma 5.

$b_{2,1}$ is false, we append the symbol σ if w is a source of G, or the symbol τ otherwise. Next, we concatenate the signature $\Sigma_{2,1}$. Once both Σ_1 and Σ_2 have been computed, we verify that each of them is short (a necessary condition by Lemma 3), otherwise we discard the candidate tuples. Finally, we set $X(\nu, \Sigma_1, \Sigma_2, b_{1,1}, b_{2,2}) = C$.

By Observation 2, we have $2^{O(k)}$ possible pairs of signatures to consider, and performing the above operations takes $O(k)$ time for each pair. ◀

The next tools will be useful for the remaining lemmas. The next result is based on the fact that face boundaries containing long sequences of non-switch vertices are irrelevant for the sake of computing the least number of saturating edges; see Figure 4 for an illustration.

▶ **Lemma 5** (⋆). *Let f be an inner face of G with n_f vertices, and let λ_f be an upward angle assignment for f with h switch-angles. The minimum number of edges that saturate all switch vertices of G forming an angle labeled $+1$ in f can be computed in $O(2^{O(h^2)} + n_f)$ time.*

▶ **Lemma 6.** *Let ν be a node of T and let μ be a child of ν. Suppose that G_ν is plane and a half-boundary B of ν contains a half-boundary B' of μ (B and B' may possibly coincide). Given an internal assignment λ of G_ν and the signature of B', the restriction of the signature of B to B' can be computed in $O(k)$ time.*

Proof. Let Σ' be the signature of B', we compute the desired signature Σ as follows. If Σ' does not contain any symbol in $\{\sigma, \tau\}$ whose corresponding vertex is bifacial, then $\Sigma = \Sigma'$. Otherwise we initialize $\Sigma = \Sigma'$ and proceed as follows. For each σ or τ whose corresponding vertex w is bifacial and incident to an inner face f of G_ν, we verify whether λ has labeled $+1$ the angle that w makes in f. If so, we replace the symbol σ or τ with σ_ℓ or τ_ℓ, respectively. By construction, Σ is the restriction of the signature of B to B'. ◀

The next result will be used to bound the number of interesting children of a P-node.

▶ **Lemma 7** (⋆). *Let ν be a P-node of T with poles u, v. Suppose that G_ν is plane, and let μ and μ' be two children of ν none of which is a Q-node, and whose corresponding edges of $\mathrm{skel}(\nu)^-$ are consecutive in the permutation fixed by the planar embedding of G_ν. Also, suppose that for both μ and μ' it holds that the pair of signatures of its two half-boundaries is empty. Let G' be the digraph obtained from G by removing all vertices of $G_{\mu'}$ except the poles u, v. Then G is a YES-instance of st-PEC if and only if G' is a YES-instance.*

We are now ready to deal with P- and R-nodes.

▶ **Lemma 8.** *Let ν be a P-node of T. The set of partial solutions of ν can be computed in $2^{O(k^2)} \cdot n$ time.*

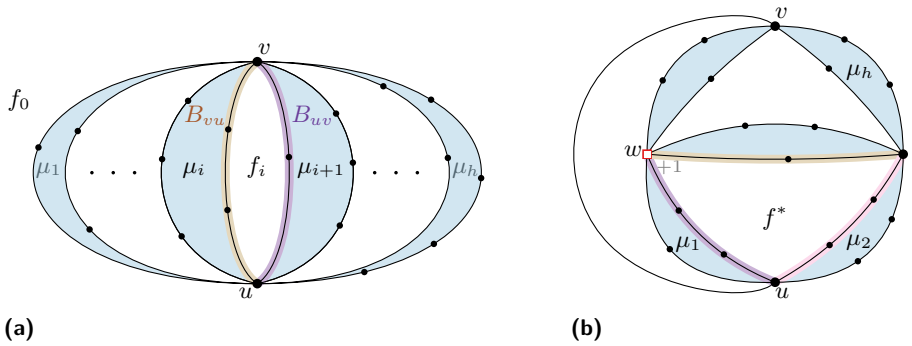


Figure 5 Illustration for the proof of (a) Lemma 8 and (b) Lemma 9.

Proof. Let u and v be the poles of ν, and let $\mu_1, \mu_2, \ldots, \mu_h$ be the $h \geq 2$ children of ν. In order to compute the partial solutions for ν, similarly as for S-nodes, we check whether sets of internal assignments of $G_{\mu_1}, G_{\mu_2}, \ldots, G_{\mu_h}$ can be combined together. For each child μ_i, let $\langle \Sigma_{1,i}, \Sigma_{2,i}, b_{1,i}, b_{2,i} \rangle$ be a candidate tuple. Let $C = \sum_{i=1}^{h} X(\mu_i, \Sigma_{1,i}, \Sigma_{2,i}, b_{1,i}, b_{2,i})$.

We first verify that $C \leq k$, and that at most one flag $b_{1,i}$ is true, as well as at most one flag $b_{2,i}$ is true. The first condition guarantees that we have not exceeded our budget k, while the second condition guarantees that the poles u, v shared by the children of ν do not receive the value $+1$ twice in the final internal assignment. If both conditions are satisfied, then we proceed as detailed below, otherwise we discard the set of candidate tuples.

Observe that h might be unbounded with respect to k, thus we cannot afford to enumerate all possible permutations of the edges of $\text{skel}^{-}(\nu)$. To overcome this issue, we make the following crucial observations. First, we know that G contains at most $2k + 2$ switches, otherwise we can safely reject the instance. Consequently, at most $2k + 2$ children of ν may contain switches different from u and v in their pertinent graphs. Second, consider now a permutation of the edges of $\text{skel}^{-}(\nu)$ and the corresponding planar embedding of G_ν. Up to a relabeling of the children, we shall assume that the half-boundary B_{vu} of μ_i and the half-boundary B_{uv} of μ_{i+1} form a face f_i of G_ν, for $i = 1, \ldots, h-1$, and that the external face f_0 of G_ν consists of B_{uv} of μ_1 and B_{vu} of μ_h; see Figure 5a. Observe now that each of u and v can contribute at most one angle labeled $+1$ and at most two angles labeled 0; all other angles at u and v must be labeled -1. Hence, besides the at most six faces in which u or v contribute an angle labeled $+1$ or 0, all other faces are such that they either contain an angle labeled $+1$, or all their angles (except those formed by u and v) are labeled 0. Therefore, the number of faces whose half-boundaries have non-empty signatures is at most $t = 2(2k + 2) + 6 = 4k + 10$ (a switch vertex may be bifacial and hence belong to two half-boundaries). Putting all together, if there exist more than t pairs that are not empty, then we can safely discard the set of candidate tuples.

Based on the previous observations, we will now assume to have at least $h - t$ empty pairs. Furthermore, if $h > 2t + 2$, at least two children are such that Lemma 7 holds for them. Consequently, removing all empty pairs except $t + 1$ preserves the existence of a solution (if any). Therefore, we shall further assume that we have $h \in O(t) \in O(k)$ pairs of signatures, and we can now enumerate all possible permutations of such pairs, and hence all possible putative planar embeddings described by $\text{skel}^{-}(\nu)$.

Consider now a fixed permutation. Following the same notation as before, assume that the half-boundary B_{vu} of μ_i and the half-boundary B_{uv} of μ_{i+1} form a face f_i of G_ν, for $i = 1, \ldots, h$. We call such faces *active*. If all values $b_{1,i}$ are true and u is a switch, we guess

whether u has an angle labeled $+1$ in some active face f_i or not. In the former case, we set flag b_1 to false and also guess which active face the angle belongs to, in the latter case we set b_1 to true. We do the same for v and flag b_2.

Next, consider a non-empty signature containing a symbol σ or τ. Let w be the vertex corresponding to that symbol. If w is not bifacial, the active face in which it forms the $+1$ angle is unique, otherwise we must guess in which of the two active faces sharing w the $+1$ angle is assigned to. After doing this procedure for all such symbols, we have exhaustively branched over the $2^{O(k)}$ angle assignments for the active faces. For each such angle assignment we can check, in $O(k)$ time, whether it is an upward assignment for each active face. If not, we discard the angle assignment, otherwise we now have an internal assignment λ of G_ν.

Next, for each active face f_i, we can apply Lemma 5 to compute the minimum number c_i of saturating edges needed to saturate all switches in f_i. Let $C + \sum_{i=1}^{h} c_i$ be the cost of the internal assignment λ. If it is larger than k, the angle assignment is discarded.

We are now ready to construct the signatures Σ_1 and Σ_2 of the half-boundaries B_{uv} and B_{vu} of ν. Since the half-boundary B_{uv} of ν coincides with B_{uv} of μ_1 (as fixed by the permutation at hand), we can invoke Lemma 6 by using λ and $\Sigma_{1,1}$ as arguments. Similarly, the signature Σ_2 is computed invoking Lemma 6 with arguments λ and $\Sigma_{2,h}$. Observe that both Σ_1 and Σ_2 are short, because $\Sigma_{1,1}$ and $\Sigma_{2,h}$ are short. Then we set $X(\Sigma_1, \Sigma_2, b_1, b_2) = \min\{X(\Sigma_1, \Sigma_2, b_1, b_2), C + \sum_{i=1}^{h} c_i\}$; taking the minimum is needed because different permutations, as well as different angle assignments of the same permutation, may yield the same pair of signatures and flags but different costs.

Putting all together, it suffices to first branch over sets of candidate tuples of size $h \in O(k)$, for each set we branch over $k^{O(k)}$ permutations, and for each permutation we further branch over the $2^{O(k)}$ possible angle assignments of the active faces. Computing the cost of an internal assignment takes $2^{O(k^2)} \cdot n$ time by using Lemma 5. ◀

▶ **Lemma 9.** *Let ν be an R-node of T. The set of partial solutions of ν can be computed in $2^{O(k^2)} \cdot n$ time.*

Proof. Let u and v be the poles of ν, and let $\mu_1, \mu_2, \ldots, \mu_h$ be the $h \geq 2$ children of ν. For each child μ_i, let $\langle \Sigma_{1,i}, \Sigma_{2,i}, b_{1,i}, b_{2,i} \rangle$ be a candidate tuple. Let $C = \sum_{i=1}^{h} X(\mu_i, \Sigma_{1,i}, \Sigma_{2,i}, b_{1,i}, b_{2,i})$.

We first verify that $C \leq k$, in order to avoid exceeding the budget. Next, we check the consistency of the flags. Recall that the vertices of skel(ν) are the poles of the children of ν. Namely, for each vertex w of skel(ν), we verify that at most one flag corresponding to it is false. If these conditions are met we proceed as detailed below, otherwise we discard the set of candidate tuples.

We now make important observations concerning the number of interesting children of ν. As in the proof of Lemma 8, we can observe that at most $2k + 2$ children of ν may contain switches different from u and v in their pertinent graphs. Now consider a child μ of ν that does not contain switches in its pertinent graph G_μ, and let u_μ and v_μ be its poles. If G admits a solution, one immediately verifies that G_μ is st-planar and its two switches are u_μ and v_μ. Consequently, in any solution, the two signatures $\Sigma_{u_\mu v_\mu}$ and $\Sigma_{v_\mu u_\mu}$ must be empty. Based on this property, it suffices to consider sets of pairs of signatures in which at most $2k + 2$ pairs are not empty.

Next, following the lines of the proof of Lemma 8, consider a non-empty signature containing a symbol σ or τ. Let w be the vertex corresponding to that symbol. If w is not bifacial, the face in which it forms the $+1$ angle is unique, otherwise we must guess in which of the two faces sharing w the $+1$ angle is assigned to. This is however not enough for R-nodes. Namely, observe that each face f^* of skel(ν)$^-$ corresponds to a face f of G_ν

whose boundary is formed by one half-boundary for each child of ν represented by an edge of f^* (which can be a real edge or a virtual edge); see Figure 5b. We call such faces *active* in the following. Moreover, the only angles that are not yet defined are those made by the vertices of skel(ν) that are switches and whose corresponding flags are all true. For these vertices we shall guess in which active face they make their $+1$ angle. Clearly, any such a vertex w belongs to multiple active faces (possibly including the external face). On the other hand, for an active face to be able to absorb a $+1$ angle, it must contain at least three angles labeled -1. Since we have at most $2k+2$ non-empty pairs, there are at most $4k+4$ active faces formed by non-empty signatures. For the other active faces, the only source of -1 angles are the vertices of skel(ν). Consequently, if w is incident to more than $4k+5$ active faces in which the number of angles labeled -1 is larger than 2, we can safely discard the set of candidate tuples. Putting all together, for each vertex w we can branch over its $O(k)$ interesting active faces to decide in which of them it will make its $+1$ angle. This procedure leads to $2^{O(k)}$ angle assignments for the active faces. For each such angle assignment we can check, in $O(k)$ time, whether it is an upward assignment for each of the active faces. If not, we discard the angle assignment, otherwise we now have an internal assignment λ of G_ν.

Next, for each active face f_i, we can apply Lemma 5 to compute the minimum number c_i of saturating edges needed to saturate all switches in f_i. Let $C + \sum_{i=1}^{h} c_i$ be the cost of the internal assignment. If it is larger than k, the angle assignment is discarded.

We are now ready to construct the signatures Σ_1 and Σ_2 of the half-boundaries B_{uv} and B_{vu} of ν. Observe that the embedding of skel(ν) if fixed up to a flipping operation, which corresponds to inverting the two signatures. Therefore, we construct Σ_1 and Σ_2 as follows. Let Σ'_i, for $i = 1, \ldots, r$ be the $r \geq 1$ signatures of the half-boundaries of the children of ν that form the half-boundary B_{uv} of ν, in the order they are encountered from u to v. Also let w_i, $i = 1, \ldots, r-1$ be the vertices of skel(ν) that belong to B_{uv}. We initialize Σ_1 with the signature obtained by invoking Lemma 6 with arguments λ and Σ'_1. For vertex w_1, we distinguish whether it is a switch of G or not. In the former case, we concatenate the symbol σ (τ) if none of its angles in G_ν is labeled as $+1$, otherwise we concatenate σ_ℓ (τ_ℓ). In the latter case, consider the two edges incident to w_1 along B_{uv}. If one edge is incoming and the other is outgoing, then we do not append any symbol. If both edges are outgoing (incoming), we append σ_ℓ (τ_ℓ). We then repeat the procedure for the remaining signatures and vertices. The signature Σ_2 is computed analogously. Once both Σ_1 and Σ_2 have been computed, we verify that each of them is short (a necessary condition by Lemma 3), otherwise we reject the set of candidate tuples. Concerning the flags, b_1 (b_2) is true if and only if all flags corresponding to u (v) are true and none of its angles in the active faces is labeled $+1$ according to λ. Finally we set $X(\Sigma_1, \Sigma_2, b_1, b_2) = \min\{X(\Sigma_1, \Sigma_2, b_1, b_2), C + \sum_{i=1}^{h} c_i\}$, as well as $X(\Sigma_2, \Sigma_1, b_2, b_1) = \min\{X(\Sigma_2, \Sigma_1, b_2, b_1), C + \sum_{i=1}^{h} c_i\}$.

Putting all together, it suffices to first branch over sets of candidate tuples of size $h \in O(k)$, for each set we branch over the $2^{O(k)}$ possible angle assignments of the active faces. Computing the cost of an internal assignment takes $2^{O(k^2)} \cdot n$ time by using Lemma 5. ◀

It remains to deal with the root ρ of T. Recall that $G_\rho = G$, and that ρ is a Q-node.

▶ **Lemma 10** (⋆). *Let G be an n-vertex biconnected digraph, let e be an edge of G, and let $k \in \mathrm{N}$. There exists an algorithm that decides, in $O(2^{O(k^2)} \cdot n)$ time, whether G can be augmented to an st-planar graph with e on its external face by adding at most k edges.*

Proof sketch. By using Lemmas 4, 8, and 9 we can traverse T bottom up until reaching the root ρ, which is the Q-node of T representing e. Following a similar procedure as for P- and R-nodes, we examine the two faces containing edge e on their boundaries and branch

over possible angle assignments. This eventually leads to an upward angle assignment of the whole graph G such that all switches, except one source and one sink in the external face, can be saturated with at most k edges (which implies that G is a positive instance), otherwise G is rejected. ◀

The proof of Theorem 1 follows by applying Lemma 10 for each of the $O(n)$ edges of G.

5 Discussion and Open Problems

We showed that st-PEC can be solved in $2^{O(k^2)} \cdot n^2$ time for biconnected digraphs. It is worth remarking that, while in principle the st-PEC problem needs not to be restricted to biconnected digraphs (for which it is already NP-hard), considering simply connected graphs would make the proof of our result more technical but not more interesting. In fact, one can simply decompose the graph into its biconnected components through a block-cutvertex tree and work with similar boundary conditions as those we already considered. More interestingly, we ask whether st-PEC belongs to the FPL (fixed parameter linear) class. On a similar note, improving the exponential function (or proving that it is asymptotically optimal under standard assumptions) would also be interesting. Lastly, it remains open whether st-PEC admits a kernel of polynomial size.

References

1 Paola Bertolazzi, Giuseppe Di Battista, and Walter Didimo. Quasi-upward planarity. *Algorithmica*, 32(3):474–506, 2002.

2 Paola Bertolazzi, Giuseppe Di Battista, Giuseppe Liotta, and Carlo Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, 12(6):476–497, 1994.

3 Carla Binucci, Emilio Di Giacomo, Giuseppe Liotta, and Alessandra Tappini. Quasi-upward planar drawings with minimum curve complexity. In *GD*, volume 12868 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 2021.

4 Pablo Burzyn, Flavia Bonomo, and Guillermo Durán. NP-completeness results for edge modification problems. *Discret. Appl. Math.*, 154(13):1824–1844, 2006. doi:10.1016/j.dam.2006.03.031.

5 Steven Chaplick, Emilio Di Giacomo, Fabrizio Frati, Robert Ganian, Chrysanthi N. Raftopoulou, and Kirill Simonov. Parameterized algorithms for upward planarity. In *SoCG*, volume 224 of *LIPIcs*, pages 26:1–26:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

6 Steven Chaplick, Emilio Di Giacomo, Fabrizio Frati, Robert Ganian, Chrysanthi N. Raftopoulou, and Kirill Simonov. Testing upward planarity of partial 2-trees. In *GD*, volume 13764 of *Lecture Notes in Computer Science*, pages 175–187. Springer, 2022.

7 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *Comput. Sci. Rev.*, 48:100556, 2023. doi:10.1016/j.cosrev.2023.100556.

8 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

9 Giuseppe Di Battista, Giuseppe Liotta, and Francesco Vargiu. Spirality and optimal orthogonal drawings. *SIAM J. Comput.*, 27(6):1764–1811, 1998.

10 Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theor. Comput. Sci.*, 61:175–198, 1988.

11 Walter Didimo, Francesco Giordano, and Giuseppe Liotta. Upward spirality and upward planarity testing. *SIAM J. Discret. Math.*, 23(4):1842–1899, 2009.

12 Walter Didimo, Michael Kaufmann, Giuseppe Liotta, and Giacomo Ortali. Rectilinear planarity of partial 2-trees. In *GD*, volume 13764 of *Lecture Notes in Computer Science*, pages 157–172. Springer, 2022.

13 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001.

14 Carsten Gutwenger and Petra Mutzel. A linear time implementation of SPQR-trees. In *GD 2000*, volume 1984 of *LNCS*, pages 77–90. Springer, 2001. doi:10.1007/3-540-44541-2.

15 John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973. doi:10.1137/0202012.

16 Liana Khazaliya, Philipp Kindermann, Giuseppe Liotta, Fabrizio Montecchiani, and Kirill Simonov. The *st*-planar edge completion problem is fixed-parameter tractable. *CoRR*, abs/2309.15454, 2023. arXiv:2309.15454.

17 Assaf Natanzon, Ron Shamir, and Roded Sharan. Complexity classification of some edge modification problems. *Discret. Appl. Math.*, 113(1):109–128, 2001. doi:10.1016/S0166-218X(00)00391-7.

18 William T. Trotter and John I. Moore Jr. The dimension of planar posets. *J. Comb. Theory, Ser. B*, 22(1):54–67, 1977.

A Combinatorial Certifying Algorithm for Linear Programming Problems with Gainfree Leontief Substitution Systems

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

Kazuhisa Makino
Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

Linear programming (LP) problems with gainfree Leontief substitution systems have been intensively studied in economics and operations research, and include the feasibility problem of a class of Horn systems, which arises in, e.g., polyhedral combinatorics and logic. This subclass of LP problems admits a strongly polynomial time algorithm, where devising such an algorithm for general LP problems is one of the major theoretical open questions in mathematical optimization and computer science. Recently, much attention has been paid to devising certifying algorithms in software engineering, since those algorithms enable one to confirm the correctness of outputs of programs with simple computations. Devising a combinatorial certifying algorithm for the feasibility of the fundamental class of Horn systems remains open for almost a decade. In this paper, we provide the first combinatorial (and strongly polynomial time) certifying algorithm for LP problems with gainfree Leontief substitution systems. As a by-product, we resolve the open question on the feasibility of the class of Horn systems.

2012 ACM Subject Classification Theory of computation → Mathematical optimization

Keywords and phrases linear programming problem, certifying algorithm, Horn system

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.47

Related Version *Full Version*: `https://arxiv.org/abs/2306.03368` [19]

Funding Partially supported by JSPS KAKENHI Grant Number JP19K22841.
Kei Kimura: Partially supported by JST, ACT-X Grant Number JPMJAX200C, Japan, and JSPS KAKENHI Grant Number JP21K17700.
Kazuhisa Makino: Partially supported by JSPS KAKENHI Grant Number JP20H05967.

1 Introduction

In this paper, we focus on linear programming (LP) problems with Leontief substitution systems. A matrix A is called *Leontief* if each column of A has at most one positive element.[1] A linear system of the form

$$A\boldsymbol{x} = \boldsymbol{b} \quad \text{and} \quad \boldsymbol{x} \geq \boldsymbol{0} \tag{1}$$

is called a *Leontief substitution system* if A is Leontief and $\boldsymbol{b}$ is nonnegative. Leontief matrices and systems were first studied in 1950s within the context of input-output analysis in economics (for which Wassily Leontief was awarded the Nobel Prize in economics in 1973; see, e.g., Leontief [21] and Dantzig [9]), and have attracted much attention in economics and operations research. There exists a line of research on algorithms for LP problems with

[1] Leontief matrices defined in this paper are sometimes called *pre-Leontief* matrices in the literature.

Leontief substitution systems; an $O(m^3 n \log n)$ strongly polynomial algorithm for a special case where A has no more than two nonzero elements in any column [1], an $O(m^2 n)$ strongly polynomial algorithm for a special case of gainfree Leontief substitution systems [16], and a simplex algorithm [4], where m and n respectively denote the number of equations and variables in (1). The gainfree property will be defined in Section 2; it intuitively says that the corresponding network, which will also be defined later, has no *gain* of flow.

We also remark that Leontief substitution systems play an important role in polyhedral combinatorics and logic. For example, Horn systems are related to Leontief substitution systems. A matrix A is called *Horn* if each row of A has at most one positive element, and a linear system $A\boldsymbol{y} \leq \boldsymbol{c}$ with Horn matrix A is called *Horn*. Thus, Horn matrices are exactly transposed Leontief matrices, and the feasibility for Horn systems coincides with that of the dual of LP problems with Leontief substitution systems. The feasibility for Horn systems was inspired by the Horn Boolean satisfiability (SAT) problem, a well-studied subclass of SAT in logic and computer science. Horn systems have been intensively studied in the literature [8, 13, 32] because they have applications in diverse areas such as logic programs, econometrics, program verification, and lattice optimization. Subclasses of Horn systems called difference constraint (DC), unit Horn, and unit-positive Horn systems are also extensively investigated, where a matrix A is *difference* if it is a $\{0, \pm 1\}$-matrix having one $+1$ and one -1 in each row [2, 11, 14, 26], *unit Horn* if it is a Horn $\{0, \pm 1\}$-matrix [5, 30], and *unit-positive Horn* if it is an integral Horn matrix with the positive elements being one [30,31][2]. We note that unit and unit-positive Horn systems are sometimes called Horn constraint and extended Horn, respectively. By definition, difference matrices are unit Horn, and unit Horn matrices are unit-positive Horn. All these matrices are transposed gainfree Leontief matrices, which will be discussed in the next section. The feasibility problem is combinatorially solvable in $O(mn)$ for DC systems [2, 11] and $O(m^2 n)$ for unit and unit-positive Horn systems [5], where m and n respectively denote the number of variables and inequalities in the system.

In this paper, we study certifying algorithms for LP problems with gainfree Leontief substitution systems. Recently, much attention has been paid to certifying algorithms in software engineering; see [22] for a survey. Intuitively, an algorithm is called *certifying* if it produces not only an answer but also a certificate with which we can easily confirm that the answer is correct. For the shortest s-t path problem with positive edge length, the potential of vertices (i.e., distances from s) is a certificate of a shortest s-t path. Certifying algorithms have great advantages in practice because many commercial programs are reported to contain bugs [22]. Certifying algorithms have been proposed for various problems in mathematical optimization and computer science [3, 6, 7, 10, 12, 17, 20, 23, 24, 27, 29] in the past few decades.

Let us briefly summarize certifying algorithms related to gainfree Leontief substitution systems. Standard LP solvers output a certificate of the optimality of an optimal solution; however, no combinatorial and strongly polynomial time algorithm for general LP problems is known and algorithms that work for special types of LP problems have been extensively studied. We first note that the well-known Bellman-Ford algorithm for the shortest path problem allowing negative edge length can be regarded as a certifying algorithm for the feasibility of DC systems. In fact, the algorithm computes a feasible solution which correspond to the potential of the associated graph G if it is feasible, and a minimal infeasible subsystem that corresponds to a negative cycle in G if it is infeasible. This result was extended to the unit-two-variable-per-inequality (UTVPI) systems, where a system is called *unit-two-variable-*

[2] The unit-positive matrices coincide with the transposes of integral gainfree Leontief matrices considered in [16], since in [16] any positive element of the matrices is assumed to be one.

per-inequality if each inequality is of the form $\pm x_i \pm x_j \leq c$ for some integer c. Miné [25] proposed a certifying algorithm for the feasibility for UTVPI systems by transforming such systems to DC systems. Therefore, the feasibility for the systems admits combinatorial $O(mn)$ certifying algorithms. Gupta [15] reported that a combinatorial certifying algorithm exists for the feasibility for unit Horn systems with nonpositivity constraints on variables, and mentioned that it is open whether the feasibility problem admits combinatorial certifying algorithms when the systems are unit Horn (without nonpositivity constraints) and unit-positive Horn [15]. For LP problems with gainfree Leontief substitution systems, Jeroslow et al. proposed a combinatorial $O(m^2n)$-time certifying algorithm when it has an optimal solution [16].

In this paper, we propose a combinatorial $O(m^3n)$-time certifying algorithm for LP problems with gainfree Leontief substitution systems when the LP problems have no optimal solution, i.e., when they are unbounded or infeasible. This together with the algorithm by Jeroslow et al. provides a combinatorial $O(m^3n)$-time fully certifying algorithm for LP problems with gainfree substitution systems. As a corollary of our result, we resolve the open problem for the feasibility for unit-positive Horn systems.

Certifying infeasibility draws much attention in, e.g., the field of logic and it was open how to make existing successive-approximation type combinatorial algorithms (e.g., [5,13,16]) certifying for a fundamental class of unit Horn systems. In those algorithms, the values of variables are iteratively updated according to the constraints and for DC systems, it is sufficient to store the previous edge (or constraint) that causes the value update of a variable to obtain a certificate of infeasibility (i.e., a negative cycle). However, in unit Horn systems, this is not enough; we have to store all the history of the value updates of the variables to certify infeasibility. Our algorithm stores in which iteration the values of variables are updated and how the values can be derived by the given constraints and utilizes these data to compute a certificate of infeasibility when the system is infeasible.

Our algorithm is based on the directed hypergraph representation of Leontief substitution systems introduced by Jeroslow et al. [16], and computes a certificate of infeasibility based on Farkas' lemma, called a Farkas' certificate, which was also used by Gupta [15] for unit Horn systems with nonpositivity constraints. Moreover, our algorithm for the dual feasibility can be seen as an extension of the Bellman-Ford algorithm for the feasibility for DC systems. In fact, if a DC system is given, then our algorithm finds a feasible solution if it is feasible, and a minimal infeasible subsystem that corresponds to a negative cycle in the associated graph if it is infeasible, which is the same as the Bellman-Ford algorithm.

The rest of the paper is organized as follows. Section 2 formally defines our problem and introduces the same directed hypergraph representation of Leontief substitution systems as in [16]. Section 3 provides our main algorithm, i.e., a combinatorial certifying algorithm for LP problems with gainfree Leontief substitution systems. Section 4 concludes the paper.

2 Preliminaries

Let $\mathbb{R}$, $\mathbb{R}_+$, and $\mathbb{R}_{++}$ denote the sets of reals, nonnegative reals, and positive reals, respectively. For positive integers m and n, a matrix $A \in \mathbb{R}^{m\times n}$ is called *Leontief* if each column contains at most one positive entry. In this paper, it is always assumed that the positive elements of A are all ones unless otherwise stated, since it is sufficient for our purpose as stated below. Let $A \in \mathbb{R}^{m\times n}$ be an $m \times n$ matrix, and let $\boldsymbol{b} \in \mathbb{R}^m$ be a vector of dimension m. A linear system of the form

$$A\boldsymbol{x} = \boldsymbol{b} \quad \text{and} \quad \boldsymbol{x} \in \mathbb{R}^n_+$$

is called a *Leontief substitution system* if A is Leontief and $\boldsymbol{b} \in \mathbb{R}^m_+$.

In this paper, we consider the following linear programming (LP) problem:

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{R}^n_+, \end{array} \tag{2}$$

where the constraint is a Leontief substitution system and $\boldsymbol{c} \in \mathbb{R}^n$. As stated above, we assume throughout the paper that the positive elements of A are all ones unless otherwise stated, since otherwise it can be obtained by scaling the variables in the LP problem (2).

We particularly focus on the subclass of LP with Leontief substitution systems satisfying the *gainfree* property. To define gainfreeness, it is convenient to introduce a directed hypergraph representation [16] of Leontief substitution systems. This representation is also used to state our algorithms.

A directed hypergraph $\mathcal{H}$ is an ordered pair $\mathcal{H} = (V, \mathcal{E})$, where V is a finite set called a vertex set and $\mathcal{E}$ is a set of hyperarcs. A hyperarc $E \in \mathcal{E}$ is an ordered pair $(H(E), T(E))$ of its head and tail sets, where $H(E), T(E) \subseteq V$ and $H(E) \cap T(E) = \emptyset$. In our use, $|H(E)|$ is always at most one, i.e., $|H(E)| \leq 1$. Hence, we denote $H(E)$ by $h(E)$, and when $|h(E)| = 1$, we identify $h(E)$ with the unique element in $h(E)$, e.g., if $v \in h(E)$, then we write $h(E) = v$.

Now, we explain how to define an associated directed hypergraph $\mathcal{H} = (V, \mathcal{E})$ from a given LP problem with a Leontief substitution system (2). For a positive integer k, let $[k] = \{1, \ldots, k\}$. Let $V = \{v_i \mid i \in [m]\}$, where v_i corresponds to the ith row of A in (2) for $i \in [m]$, and let $\mathcal{E} = \{E_j \mid j \in [n]\}$, where for each $j \in [n]$ a hyperarc E_j is defined as $h(E_j) = v_i$ if $A_{ij} = 1$ for some $i \in [m]$ and $h(E_j) = \emptyset$ otherwise (i.e., $A_{ij} \leq 0$ for all $i \in [m]$), and $T(E_j) = \{v_i \in V \mid A_{ij} < 0\}$. Note that for each $j \in [n]$ hyperarc E_j corresponds to variable x_j in (2). We also associate a length function $\ell : \mathcal{E} \to \mathbb{R}$ to the hyperarc set $\mathcal{E}$, where $\ell(E_j) = c_j$ for each $E_j \in \mathcal{E}$. Moreover, we associate a positive value to each element of the tails of the hyperarcs in $\mathcal{E}$, namely, $\gamma : \bigcup_{j\in[n]}(\{E_j\} \times T(E_j)) \to \mathbb{R}_{++}$ defined as $\gamma(E_j, v_i) = -A_{ij}\ (> 0)$ for each $E_j \in \mathcal{E}$ and $v_i \in T(E_j)$. Note that the directed hypergraph is defined by matrix A and vector $\boldsymbol{c}$ (and $\boldsymbol{b}$ is irrelevant).

▶ **Example 1.** For the following input data, the associated directed hypergraph is drawn in Figure 1.

$$A = \begin{pmatrix} -(1/2) & 0 & 1 & 1 & 0 \\ 1 & -(1/3) & 0 & 0 & 0 \\ 0 & 1 & -9 & 0 & 1 \\ -(1/3) & -3 & -1 & 0 & 0 \end{pmatrix} \text{ and } c = \begin{pmatrix} -6 \\ 5 \\ 3 \\ -4 \\ 2 \end{pmatrix}. \tag{3}$$

A *directed path* in directed hypergraph $\mathcal{H}$ from vertex v_1 to v_{k+1} is defined by a nonempty sequence $v_1E_1v_2E_2v_3 \cdots E_kv_{k+1}$, with no intermediate vertex or hyperarc repeated, such that $v_{i+1} = h(E_i)$ and $v_i \in T(E_i)$ for $i = 1, \ldots, k$. A directed path from vertex v_1 to v_{k+1} is a *directed cycle* if $v_1 = v_{k+1}$.

Now, we are ready to define gainfreeness.

▶ **Definition 2** (Gainfreeness). *Let $v_1E_1v_2E_2v_3 \cdots E_kv_{k+1}$ be a directed cycle, where $v_1 = v_{k+1}$. The gain of this directed cycle is defined by $1/\prod_{i=1}^{k}\gamma(E_i, v_i)$. We term a Leontief substitution system (and its defining matrix)* gainfree *if the gain of every directed cycle in the associated directed hypergraph is at most one.*

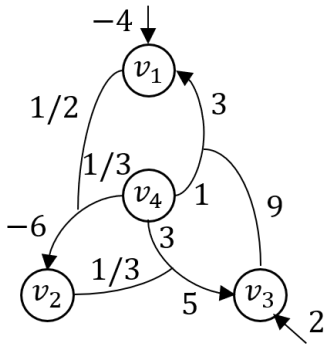


Figure 1 The directed hypergraph representation corresponding to the input (3).

From definition, unit and unit-positive Horn matrices are transpose of gainfree Leontief matrices.

▶ **Example 3.** In Example 1, the unique directed cycle of the directed hypergraph representation is $v_1E_1v_2E_2v_3E_3v_1$, where each E_i corresponds to the ith column of A. The gain of this cycle is $1/(1/2 \cdot 1/3 \cdot 9) = 2/3 \leq 1$. Hence, matrix A in (3) is gainfree.

Now, we recall some notion from LP theory. A vector $\boldsymbol{x} \in \mathbb{R}^n_+$ is called a *feasible solution* of (2) if it satisfies the constraints in (2). An LP problem is *feasible* if it has a feasible solution, and *infeasible* otherwise. A vector $\boldsymbol{x} \in \mathbb{R}^n_+$ is called an *optimal solution* of (2) if it is feasible and $\boldsymbol{c}^T\boldsymbol{x} \leq \boldsymbol{c}^T\boldsymbol{x}'$ for any feasible solution $\boldsymbol{x}'$. When an LP problem has an optimal solution $\boldsymbol{x}$, the objective value $\boldsymbol{c}^T\boldsymbol{x}$ is called an *optimal value*. An LP problem is either feasible or infeasible, and when it is feasible either it has an optimal solution or it is unbounded (i.e., its optimal value is not bounded below). Since we consider certifying algorithms, we have to produce a certificate in each case. To state what constitutes a certificate in each case, we recall the *dual* LP problem of (2):

$$\begin{array}{ll} \text{maximize} & \boldsymbol{y}^T\boldsymbol{b} \\ \text{subject to} & \boldsymbol{y}^TA \leq \boldsymbol{c}^T \\ & \boldsymbol{y} \in \mathbb{R}^m. \end{array} \tag{4}$$

To contrast, the LP problem (2) is sometimes called the *primal* LP problem in what follows.

The following duality theorem of LP is well-known.

▶ **Theorem 4** (E.g., [28]). *For the LP problem* (2) *and its dual problem* (4)*, exactly one of the following holds:*

- **(i)** *both* (2) *and* (4) *have feasible solutions whose objective values are the same,*
- **(ii)** (2) *is infeasible, and* (4) *feasible and unbounded,*
- **(iii)** (2) *is feasible and unbounded, and* (4) *is infeasible;*
- **(iv)** *both* (2) *and* (4) *are infeasible.*

We regard a feasible solution as a certificate of feasibility of an LP problem. For infeasibility we use the following lemma.

▶ **Lemma 5** (E.g., [28]). *The LP problem* (2) *is infeasible if and only if*

$$\boldsymbol{z}^TA \leq \boldsymbol{0},\ \boldsymbol{z}^T\boldsymbol{b} > 0,\ \textit{and}\ \boldsymbol{z} \in \mathbb{R}^m \tag{5}$$

is feasible. Moreover, the dual LP problem (4) *is infeasible if and only if*

$$A\boldsymbol{r} = \boldsymbol{0},\ \boldsymbol{c}^T\boldsymbol{r} < 0,\ \textit{and}\ \boldsymbol{r} \in \mathbb{R}^n_+ \tag{6}$$

is feasible.

Now, we define what constitute certificates for the four possible cases in Theorem 4.

(i) Feasible solutions of (2) and (4) whose objective values are the same,
(ii) a feasible solution of (5) (called a *Farkas' certificate* of infeasibility of (2)) and a feasible solution of (4),
(iii) a feasible solution of (2) and a feasible solution of (6) (called a *Farkas' certificate* of infeasibility of (4)),
(iv) a feasible solution of (5) and a feasible solution of (6).

With those certificates, we can confirm the correctness of the output of our certifying algorithm for solving the LP problem (2) by checking if given vectors satisfy the corresponding linear systems. We note that for case (ii) (resp., (iii)) a feasible solution of (5) (resp., (6)) is a direction of unboundedness.

3 Main algorithms

In this section, we provide a combinatorial certifying algorithm for LP problems with gainfree Leontief substitution systems (2) and show the following theorem. Here, a combinatorial algorithm consists only of additions, subtractions, multiplications, and comparisons. Recall that m is the number of constraints and n is the number of variables in (2).

▶ **Theorem 6** (Main). *The LP problems with gainfree Leontief substitution systems* (2) *admit a combinatorial strongly polynomial time certifying algorithm that runs in* $O(m^3 n)$ *time.*

Our algorithm extends the non-certifying algorithm in [16]. Let us first summarize the algorithm in [16], which consists of ValueIteration and PrimalRetrieval. ValueIteration determines feasibility of the dual LP problem (4). It starts from a sufficiently large vector and iteratively compute an upper bound of the value of each variable derived from the constraints in (4). For an LP problem with a gainfree Leontief substitution system, m iterations are shown to be sufficient to obtain a feasible solution if the dual LP problem is feasible. Then, feasibility of the primal LP problem (2) can be determined using the data computed in ValueIteration, and when both primal and dual LP problems are feasible, PrimalRetrieval computes a feasible solution of the primal LP problem. This algorithm outputs feasible solutions of the primal and dual LP problems with the same objective values as a certificate of primal and dual feasibility for case (i) in Theorem 4 in Section 2.

To make the algorithm in [16] also certifying for primal and dual infeasibility (i.e., for cases (ii-iv) in Theorem 4), we modify the algorithm and add several subroutines to it. We first modify ValueIteration to DualFeasibility (Algorithm 2). In DualFeasibility, when the upper bound $\boldsymbol{y}^{(k)}$ for the dual variables is updated in the kth iteration of the for-loop starting from line 2, we store (i) variables changed in the iteration in array change$^{(k)}$ and (ii) vectors $\boldsymbol{r}^{(k)}$ that represents how an upper bound $\boldsymbol{y}^{(k)}$ is derived from the constraint in (4). This enables us to compute a Farkas' certificate of dual infeasibility in FarkasCertificateOfDualInfeasibility (Algorithm 4) when the dual LP problem is infeasible. This modification also makes our algorithm different from the one in [15]. Since the upper bound $\boldsymbol{y}^{(m)}$ computed in DualFeasibility contains symbol M as described below, we compute in DualSolution (Algorithm 3) a feasible solution of the dual LP problem from $\boldsymbol{y}^{(m)}$ when the dual LP problem is feasible. Intuitively, if we substitute sufficiently large value for M, then $\boldsymbol{y}^{(m)}$ becomes a feasible solution. Then PrimalFeasibility (Algorithm 6) determines the feasibility of the primal LP problem (2) using the same criterion as in (ii) of Theorem 3.6 in [16]. PrimalSolution is almost the same as PrimalRetrieval in [16], however,

the former computes a primal feasible solution even when the dual LP problem is infeasible by running DualFeasibility for a feasible dual LP problem where $\boldsymbol{c}$ is set to $\mathbf{0}$. Finally, in DualFeasibility we treat M as a symbol representing an "arbitrary large" number so that we can compute a Farkas' certificate of primal infeasibility in FarkasCertificateOfPrimalInfeasibility (Algorithm 8) by just taking the coefficient of M in $\boldsymbol{y}^{(m)}$. More precisely, for any real numbers $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathbb{R}$, we define $\alpha_1 M + \beta_1 > \alpha_2 M + \beta_2$ if and only if $\alpha_1 > \alpha_2$ or ($\alpha_1 = \alpha_2$ and $\beta_1 > \beta_2$)[3]. In what follows, we denote by e_i (resp., e_E) an unit vector of appropriate size, where its ith element (resp., its element indexed by hyperarc E) is 1 and all other elements are 0.

For the readability, we first describe a certifying algorithm for the feasibility for the dual of the LP problems (with gainfree Leontief substitution systems) in Subsection 3.1 and one for the feasibility for the primal LP problems in Subsection 3.2. A proof of Theorem 6 will be given in Subsection 3.3. Due to the space limitation, we omit proofs of most results.

3.1 A certifying algorithm for the feasibility for the dual LP problem

In this subsection, we provide a certifying algorithm for the feasibility for the dual (4) of the LP problem with a gainfree Leontief substitution system. The main algorithm (Algorithm 1) first calls subroutine DualFeasibility (Algorithm 2), which determines the feasibility of the dual LP problem (4). If it is feasible, then subroutine DualSolution (Algorithm 3) is called to compute a feasible solution of the dual LP problem; otherwise, subroutine FarkasCertificateOfDualInfeasibility (Algorithm 4) is called to compute a Farkas' certificate of dual infeasibility.

Algorithm 1 A combinatorial certifying algorithm for the feasibility for the dual of the LP problems with gainfree Leontief substitution systems.

Input: A matrix A and a vector $\boldsymbol{c}$ for the constraint of the dual LP problem (4).
1 $(\boldsymbol{y}^{(m)}, \boldsymbol{r}^{(k)}(k = 0, ..., m), \text{change}^{(k)}(k = 0, ..., m), p^{(k)}(k = 0, ..., m), \text{nontriv}^{(m)}, \boldsymbol{q}, \text{VALUE}) \leftarrow$ DualFeasibility$(A, \boldsymbol{c})$.
2 **if** VALUE $=$ true **then**
3 $\quad \boldsymbol{y}^* \leftarrow$ DualSolution$(A, \boldsymbol{c}, \boldsymbol{y}^{(m)})$.
4 $\quad$ **print** "dual-feasible" and **return** $\boldsymbol{y}^*$.
5 **else**
6 $\quad \boldsymbol{r}^* \leftarrow$ FarkasCertificateOfDualInfeasibility$(A, \boldsymbol{c}, \boldsymbol{y}^{(m)}, \boldsymbol{r}^{(k)}(k = 0, ..., m), \text{change}^{(k)}(k = 0, ..., m), p^{(k)}(k = 0, ..., m))$.
7 $\quad$ **print** "dual-infeasible" and **return** $\boldsymbol{r}^*$.
8 **end**

Before going into proofs of correctness of these algorithms, we show an example how these algorithms work. We only describe how upper bound $\boldsymbol{y}^{(k)}$ and vector $\boldsymbol{r}_v^{(k)}$ are updated in each iteration of the for-loop starting from line 2 in DualFeasibility in the example for readability. Also, we omit the input vector $\boldsymbol{b}$ in the example, since $\boldsymbol{b}$ is irrelevant to feasibility of the dual LP problem (4).

[3] We may regard $\alpha M + \beta$ as an element (α, β) of $\mathbb{R}^2$ equipped with a lexicographical order, i.e., $(\alpha_1, \beta_1) > (\alpha_2, \beta_2)$ if and only if $\alpha_1 > \alpha_2$ or ($\alpha_1 = \alpha_2$ and $\beta_1 > \beta_2$). This fact was pointed out by a reviewer. We use notation $\alpha M + \beta$, since we substitute some value for M in our algorithm.

Algorithm 2 DualFeasibility.

Input: A matrix A and a vector $\boldsymbol{c}$ for the constraint of the dual LP problem (4).

For each $v \in V$, $y^{(0)}(v) \leftarrow M$, $\boldsymbol{r}_v^{(0)} \leftarrow \mathbf{0}$, change$^{(0)}(v) \leftarrow$ false, $p^{(0)}(v) \leftarrow \emptyset$, nontriv$^{(0)}(v) \leftarrow$ false, and $q(v) \leftarrow 0$.
for $k = 1, \ldots, m$ **do**
  for $v \in V$ **do**
    if $y^{(k-1)}(v) > \min\left\{\ell(E) + \sum_{u \in T(E)} \gamma(E,u) y^{(k-1)}(u) \mid E \in \mathcal{E}, h(E) = v\right\}$ **then**
      Choose an arbitrary $E \in \operatorname{argmin}\left\{\ell(E) + \sum_{u \in T(E)} \gamma(E,u) y^{(k-1)}(u) \mid E \in \mathcal{E}, h(E) = v\right\}$.
      $y^{(k)}(v) \leftarrow \ell(E) + \sum_{u \in T(E)} \gamma(E,u) y^{(k-1)}(u)$.
      $p^{(k)}(v) \leftarrow E$.
      $\boldsymbol{r}_v^{(k)} \leftarrow e_E + \sum_{u \in T(E)} \gamma(E,u) \boldsymbol{r}_u^{(k-1)}$.
      change$^{(k)}(v) \leftarrow$ true.
      if *for every* $u \in T(E)$ nontriv$^{(k-1)}(u) =$ true *(this includes the case that* $T(E) = \emptyset$*)* **then**
        nontriv$^{(k)}(v) \leftarrow$ true and $q(v) \leftarrow k$.
      else
        nontriv$^{(k)}(v) \leftarrow$ nontriv$^{(k-1)}(v)$.
      end
    else
      $y^{(k)}(v) \leftarrow y^{(k-1)}(v)$, $p^{(k)}(v) \leftarrow \emptyset$, $\boldsymbol{r}_v^{(k)} \leftarrow \boldsymbol{r}_v^{(k-1)}$, change$^{(k)}(v) \leftarrow$ false, and nontriv$^{(k)}(v) \leftarrow$ nontriv$^{(k-1)}(v)$.
    end
  end
end
if $y^{(m)}(v) > \min\left\{\ell(E) + \sum_{u \in T(E)} \gamma(E,u) y^{(m)}(u) \mid E \in \mathcal{E}, h(E) = v\right\}$ for some $v \in V$ **then**
  VALUE $\leftarrow$ false.
else if $0 > \ell(E) + \sum_{u \in T(E)} \gamma(E,u) y^{(m)}(u)$ *for some* $E \in \mathcal{E}$ *with* $h(E) = \emptyset$ **then**
  VALUE $\leftarrow$ false.
else
  VALUE $\leftarrow$ true.
end
return $(\boldsymbol{y}^{(m)}, \boldsymbol{r}^{(k)}(k = 0, ..., m), \text{change}^{(k)}(k = 0, ..., m), p^{(k)}(k = 0, ..., m), \text{nontriv}^{(m)}, \boldsymbol{q}, \text{VALUE})$.

▶ **Example 7.** For the following matrix A (whose transpose is unit Horn) and vector $\boldsymbol{c}$

$$A = \begin{pmatrix} -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 1 & 0 \\ -1 & -1 & -1 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } \boldsymbol{c} = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

Algorithm 3 DualSolution.

Input: A matrix A and a vector $\boldsymbol{c}$ for the constraint of the dual LP problem (4), and an n-dimensional vector $\boldsymbol{y}^{(m)}$ with each entry being a linear function of M.

for *each* $E \in \mathcal{E}$ **do**
  Define two integers $\alpha(E)$ and $\beta(E)$ such that $\alpha(E)M + \beta(E) = y^{(m)}(h(E)) - \ell(E) - \sum_{u \in T(E)} \gamma(E,u) y^{(m)}(u)$, if where we define $y^{(m)}(\emptyset) = 0$.
end
if *all* $E \in \mathcal{E}$ *satisfy* $\alpha(E) \geq 0$ **then**
  $\lambda \leftarrow 0$.
else
  $\lambda \leftarrow \max\left\{ \frac{\beta(E)}{-\alpha(E)} \mid E \in \mathcal{E}, \alpha(E) < 0 \right\}$.
end
Let $\boldsymbol{y}^*$ be the vector obtained from $\boldsymbol{y}^{(m)}$ by substituting λ for M.
return $\boldsymbol{y}^*$.

$\boldsymbol{y}^{(0)} = (M, M, M, M)^T$ and $\boldsymbol{r}_{v_i}^{(0)} = \mathbf{0}$ for $i = 1, 2, 3, 4$.
Iteration 1: $\boldsymbol{y}^{(1)} = (0, 0, 0, 0)^T$ and $\boldsymbol{r}_{v_i}^{(1)} = e_{i+3}$ $(i = 1, 2, 3, 4)$.
Iteration 2: $\boldsymbol{y}^{(2)} = (0, -1, 0, 0)^T$ and $\boldsymbol{r}_{v_2}^{(2)} = e_1 + e_4 + e_7$.
Iteration 3: $\boldsymbol{y}^{(3)} = (0, -1, -1, 0)^T$ and $\boldsymbol{r}_{v_3}^{(3)} = e_1 + e_2 + e_4 + 2e_7$.
Iteration 4: $\boldsymbol{y}^{(4)} = (-1, -1, -1, 0)^T$ and $\boldsymbol{r}_{v_1}^{(4)} = e_1 + e_2 + e_3 + e_4 + 3e_7$.
Now, the first inequality is violated by $\boldsymbol{y}^{(4)}$ as $(-1, 0, 1, -1)\boldsymbol{y}^{(4)} = 0 > -1$. Then, by running FarkasCertificateOfDualInfeasibility, we have $\boldsymbol{r}^* = \boldsymbol{r}_{v_2}^{(5)} - \boldsymbol{r}_{v_2}^{(2)} = e_1 + e_2 + e_3 + 3e_7$. Here, $A\boldsymbol{r}^* = (0, 0, 0, 0)^T$ and $c^T\boldsymbol{r}^* = -1$. Hence, $\boldsymbol{r}^*$ is a Farkas' certificate of infeasibility of the dual LP problem (4).

In the remainder of this subsection, we will prove correctness of Algorithm 1. We show correctness of subroutines DualFeasibility, DualSolution[4], and FarkasCertificateOfDualInfeasibility, and show the following proposition.

▶ **Proposition 8.** *Algorithm 1 is a combinatorial strongly polynomial time certifying algorithm that runs in* $O(m^3 n)$ *time for the feasibility for the dual* (4) *of the LP problem with a gainfree Leontief substitution system.*

To show Proposition 8, we first deal with the case where Algorithm 1 prints "dual-feasible" (or, equivalently, DualFeasibility returns true) in Lemma 9 below. Then, we deal with the case where Algorithm 1 prints "dual-infeasible" (or, equivalently, DualFeasibility returns false) in Lemma 10 below.

▶ **Lemma 9.** *If* DualFeasibility *returns* true*, then the dual LP problem* (4) *is feasible and* DualSolution *outputs a feasible solution to it.*

Proof. We show that the output $\boldsymbol{y}^*$ of DualSolution is a feasible solution of the dual LP problem (4). We divide the proof into cases according to the conditions in the definition of λ in DualSolution.

[4] DualSolution uses a division, however, we can avoid the division by using ValueIteration in [16] to obtain a feasible dual solution.

Algorithm 4 FarkasCertificateOfDualInfeasibility.

Input: A matrix A and a vector $\boldsymbol{c}$ for the constraint of the dual LP problem (2), $\boldsymbol{y}^{(m)}$, and $\boldsymbol{r}^{(k)}$, change$^{(k)}$, and $p^{(k)}$ for $k = 0, ..., m$.

1 **if** $y^{(m)}(v) > \min\left\{\ell(E) + \sum_{u\in T(E)} \gamma(E,u)y^{(m)}(u) \mid E \in \mathcal{E}, h(E) = v\right\}$ for some $v \in V$ **then**

2 Choose one $v \in V$ such that $y^{(m)}(v) > \min\left\{\ell(E) + \sum_{u\in T(E)} \gamma(E,u)y^{(m)}(u) \mid E \in \mathcal{E}, h(E) = v\right\}$.

3 Choose an arbitrary $E \in \mathcal{E}$ with $h(E) = v$ that minimizes $\ell(E) + \sum_{u\in T(E)} \gamma(E,u)y^{(m)}(u)$.

4 $w_{m+1} \leftarrow v$.

5 $\boldsymbol{r}^{(m+1)}_{w_{m+1}} \leftarrow e_E + \sum_{u\in T(E)} \gamma(E,u)\boldsymbol{r}^{(m)}_u$.

6 $E^{(m+1)} \leftarrow E$.

7 /* Find a cycle */

8 **for** $k = m+1, \ldots, 2$ **do**

9 Choose an arbitrary $u \in T(E^{(k)})$ such that change$^{(k-1)}(u) =$ true.

10 $w_{k-1} \leftarrow u$.

11 $E^{(k-1)} \leftarrow p^{(k-1)}(w_{k-1})$.

12 **if** $w_{k-1} = w_q$ *for some* $q \geq k$ **then**

13 $t \leftarrow q$.

14 $s \leftarrow k-1$.

15 Break.

16 **end**

17 **end**

18 $\boldsymbol{r}^* \leftarrow \boldsymbol{r}^{(t)}_{w_t} - \boldsymbol{r}^{(s)}_{w_s}$.

19 **return** $\boldsymbol{r}^*$.

20 **else**

21 Choose one $E \in \mathcal{E}$ with $h(E) = \emptyset$ such that $0 > \ell(E) + \sum_{u\in T(E)} \gamma(E,u)y^{(m)}(u)$.

22 $\boldsymbol{r}^* \leftarrow \boldsymbol{e}_E + \sum_{u\in T(E)} \gamma(E,u)\boldsymbol{r}^{(m)}_u$.

23 **return** $\boldsymbol{r}^*$.

24 **end**

Fix $E \in \mathcal{E}$. Note that we have

$$y^{(m)}(h(E)) \leq \ell(E) + \sum_{u\in T(E)} \gamma(E,u)y^{(m)}(u),$$

since the conditions of "if " and "else if " in lines 20 and 22, respectively, are false in DualFeasibility, where we define $y^{(m)}(\emptyset) = 0$. Hence, we have $\alpha(E)M + \beta(E) \leq 0$. It follows that $\alpha(E) \leq 0$. If $\alpha(E) = 0$, then $\beta(E) \leq 0$ and $\boldsymbol{y}^*$ satisfy the constraint in the dual LP problem (4) corresponding to E. If $\alpha(E) < 0$, then $\boldsymbol{y}^*$ also satisfies the inequality in the dual LP problem (4) corresponding to E, since $\lambda \geq \frac{\beta(E)}{-\alpha(E)}$ by definition. This completes the proof. ◀

Next, we treat the case where DualFeasibility returns false and show the following.

▶ **Lemma 10.** *If* DualFeasibility *returns* false*, then the dual LP problem* (4) *is infeasible and* FarkasCertificateOfDualInfeasibility *returns a Farkas' certificate of dual infeasibility.*

The proof of Lemma 10 is the most technical part of our results. Intuitively, when DualFeasibility returns false, we can find a "negative cycle" as in the case of difference constraint (DC) systems. Here, the gainfree property ensures that such a negative cycle, together with paths to the tails of hyperarcs in the cycle, corresponds to an infeasible subsystem of (4). The vectors $\boldsymbol{r}_v^{(k)}$ store how the negative cycle is derived from constraints in (4) and help to compute such a subsystem (with multiplicity) in FarkasCertificateOfDualInfeasibility.

We first treat the case where the "if " condition in line 20 is false and the "else if " condition in line 22 is true in DualFeasibility.

▶ **Lemma 11.** *If* DualFeasibility *returns* false *as the "if " condition in line 20 is false and the "else if " condition in line 22 is true, then the dual LP problem* (4) *is infeasible and* FarkasCertificateOfDualInfeasibility *returns a Farkas' certificate of dual infeasibility.*

To show this lemma, we need some auxiliary claims, which can be shown by mathematical induction on k.

▷ Claim 12. In the end of DualFeasibility, for all $k \in \{1, \dots, m\}$ and $v \in V$, $y^{(k)}(v)$ contains M if and only if $\mathrm{nontriv}^{(k)}(v) =$ false. Moreover, if $y^{(k)}(v)$ contains M, the coefficient of M is positive for all $k = 1, \dots, m$ and $v \in V$.

▷ Claim 13. In the end of DualFeasibility, for all $k \in \{1, \dots, m\}$ and $v \in V$, we have $A\boldsymbol{r}_v^{(k)} \leq \boldsymbol{e}_v$, $\boldsymbol{r}_v^{(k)} \geq \boldsymbol{0}$, and $\boldsymbol{c}^T\boldsymbol{r}_v^{(k)}$ equals the constant term of $y^{(k)}(v)$. If $\mathrm{nontriv}^{(k)}(v) =$ true, then $A\boldsymbol{r}_v^{(k)} = \boldsymbol{e}_v$ and $\boldsymbol{c}^T\boldsymbol{r}_v^{(k)} = y^{(k)}(v)$.

Proof of Lemma 11. We show that $\boldsymbol{r}^*$ returned by FarkasCertificateOfDualInfeasibility is actually a Farkas' certificate of dual infeasibility, i.e., (i) $\boldsymbol{r}^* \geq \boldsymbol{0}$, (ii) $A\boldsymbol{r}^* = \boldsymbol{0}$, and (iii) $\boldsymbol{c}^T\boldsymbol{r}^* < 0$ (see Lemma 5).

For (i), from Claim 13, we have that $\boldsymbol{r}^* (= \boldsymbol{e}_E + \sum_{u \in T(E)} \gamma(E,u)\boldsymbol{r}_u^{(m)})$ is a sum of nonnegative vectors. Hence, $\boldsymbol{r}^* \geq \boldsymbol{0}$.

For (ii), observe that to satisfy $0 > \ell(E) + \sum_{u \in T(E)} \gamma(E,u) y^{(m)}(u)$, $y^{(m)}(u)$ must not contain M for each $u \in T(E)$, since otherwise the right-hand side of the inequality contains M with a positive coefficient from Claim 12 and thus greater than zero. Hence, for each $u \in T(E)$ $\mathrm{nontriv}^{(m)}(u) =$ true from Claim 12, implying that $A\boldsymbol{r}_u^{(m)} = \boldsymbol{e}_u$ from Claim 13. Therefore, we have

$$A\boldsymbol{r}^* = A\boldsymbol{e}_E + \sum_{u \in T(E)} \gamma(E,u) A\boldsymbol{r}_u^{(m)} = -\sum_{u \in T(E)} \gamma(E,u)\boldsymbol{e}_u + \sum_{u \in T(E)} \gamma(E,u)\boldsymbol{e}_u = 0.$$

For (iii), for each $u \in T(E)$ we have $\boldsymbol{c}^T\boldsymbol{r}_u^{(m)} = y^{(m)}(u)$ from Claim 13 since $\mathrm{nontriv}^{(m)}(u) =$ true as shown above. Hence, we have $\boldsymbol{c}^T\boldsymbol{r}^* = \boldsymbol{c}^T\boldsymbol{e}_E + \sum_{u \in T(E)} \gamma(E,u)\boldsymbol{c}^T\boldsymbol{r}_u^{(m)} = \ell(E) + \sum_{u \in T(E)} \gamma(E,u) y^{(m)}(u) < 0$.

Therefore, $\boldsymbol{r}^*$ is a Farkas' certificate of dual infeasibility and by Lemma 5 the dual LP problem (4) is infeasible. ◀

We then deal with the case where the "if " condition in line 20 is true in DualFeasibility.

▶ **Lemma 14.** *If* DualFeasibility *returns* false *as the "if " condition in line 20 is true, then the dual LP problem* (4) *is infeasible and* FarkasCertificateOfDualInfeasibility *returns a Farkas' certificate of the dual infeasibility.*

To show Lemma 14, we need further auxiliary claims.

▷ Claim 15. In FarkasCertificateOfDualInfeasibility, for each $k = m+1, m, \ldots, s+1$, there exists $u \in T(E^{(k)})$ such that $\text{change}^{(k-1)}(u) = \text{true}$.

Claim 15 implies the following claim.

▷ Claim 16. In FarkasCertificateOfDualInfeasibility, we can always obtain a cycle.

The following claim uses the gainfree property of the LP problem (2).

▷ Claim 17. In the end of FarkasCertificateOfDualInfeasibility, for any $s+1 \leq k \leq t$ and any $u \in T(E^{(k)}) \setminus \{w_{k-1}\}$, we have $\text{nontriv}^{(k-1)}(u) = \text{true}$.

Now, we are ready to prove Lemma 14.

Proof of Lemma 14. We show that $\boldsymbol{r}^*$ is actually a Farkas' certificate of dual infeasibility, i.e., (i) $\boldsymbol{r}^* \geq \boldsymbol{0}$, (ii) $A\boldsymbol{r}^* = \boldsymbol{0}$, and (iii) $\boldsymbol{c}^T\boldsymbol{r}^* < 0$. Due to page limitation, we only prove (ii). For (ii), recall that for any $s+1 \leq k \leq t$ and any $u \in T(E^{(k)}) \setminus \{w_{k-1}\}$, we have $A\boldsymbol{r}_u^{(k-1)} = \boldsymbol{e}_u$ from Claims 13 and 17. Moreover, we have $A\boldsymbol{e}_{E^{(k)}} = \boldsymbol{e}_{h(E^{(k)})} - \sum_{u \in T(E^{(k)})} \gamma(E^{(k)}, u)\boldsymbol{e}_u$. Hence, for each $s+1 \leq k \leq t$,

$$\begin{aligned}
A\boldsymbol{r}_{w_k}^{(k)} &= A(e_{E^{(k)}} + \sum_{u \in T(E^{(k)})} \gamma(E^{(k)}, u)\boldsymbol{r}_u^{(k-1)}) \\
&= \boldsymbol{e}_{h(E^{(k)})} - \sum_{u \in T(E^{(k)})} \gamma(E^{(k)}, u)\boldsymbol{e}_u + A(\sum_{u \in T(E^{(k)})} \gamma(E^{(k)}, u)\boldsymbol{r}_u^{(k-1)}) \\
&= \boldsymbol{e}_{w_k} + \sum_{u \in T(E^{(k)})} \gamma(E^{(k)}, u)(A\boldsymbol{r}_u^{(k-1)} - \boldsymbol{e}_u) \\
&= \boldsymbol{e}_{w_k} + \sum_{u \in T(E^{(k)}) \setminus \{w_{k-1}\}} \gamma(E^{(k)}, u)(A\boldsymbol{r}_u^{(k-1)} - \boldsymbol{e}_u) \\
&\quad + \gamma(E^{(k)}, w_{k-1})(A\boldsymbol{r}_{k-1}^{(w_{k-1})} - \boldsymbol{e}_{w_k-1}) \\
&= \boldsymbol{e}_{w_k} + \gamma(E^{(k)}, w_{k-1})(A\boldsymbol{r}_{k-1}^{(w_{k-1})} - \boldsymbol{e}_{w_k-1}).
\end{aligned}$$

Namely, we have $A\boldsymbol{r}_{w_k}^{(k)} - \boldsymbol{e}_{w_k} = \gamma(E^{(k)}, w_{k-1})(A\boldsymbol{r}_{k-1}^{(w_{k-1})} - \boldsymbol{e}_{w_k-1})$. Therefore, we have

$$A\boldsymbol{r}_{w_t}^{(t)} - \boldsymbol{e}_{w_t} = \gamma(E^{(t)}, w_{t-1})(A\boldsymbol{r}_{w_{t-1}}^{(t-1)} - \boldsymbol{e}_{w_t-1}) = \cdots = \prod_{k=s+1}^{t} \gamma(E^{(k)}, w_{k-1})(A\boldsymbol{r}_{w_s}^{(s)} - \boldsymbol{e}_{w_s}).$$

Hence, we have

$$\begin{aligned}
A\boldsymbol{r}^* &= A(\boldsymbol{r}_{w_t}^{(t)} - \boldsymbol{r}_{w_s}^{(s)}) \\
&= \boldsymbol{e}_{w_t} + \prod_{k=s+1}^{t} \gamma(E^{(k)}, w_{k-1})(A\boldsymbol{r}_{w_s}^{(s)} - \boldsymbol{e}_{w_s}) - A\boldsymbol{r}_{w_s}^{(s)} \\
&= \boldsymbol{e}_{w_t} + \left(\prod_{k=s+1}^{t} \gamma(E^{(k)}, w_{k-1}) - 1\right) A\boldsymbol{r}_{w_s}^{(s)} - \prod_{k=s+1}^{t} \gamma(E^{(k)}, w_{k-1})\boldsymbol{e}_{w_s}.
\end{aligned}$$

Now, if $\text{nontriv}^{(t)}(w_t) = \text{true}$, we can show that $\text{nontriv}^{(s)}(w_s) = \text{true}$. Hence, $A\boldsymbol{r}_{w_s}^{(s)} = \boldsymbol{e}_{w_s}$ by Claim 13. Therefore, we have

$$\begin{aligned}
&\boldsymbol{e}_{w_t} + \left(\prod_{k=s+1}^{t} \gamma(E^{(k)}, w_{k-1}) - 1 \right) A\boldsymbol{r}_{w_s}^{(s)} - \prod_{k=s+1}^{t} \gamma(E^{(k)}, w_{k-1}) \boldsymbol{e}_{w_s} \\
&= \boldsymbol{e}_{w_t} + (\prod_{k=s+1}^{t} \gamma(E^{(k)}, w_{k-1}) - 1) \boldsymbol{e}_{w_s} - \prod_{k=s+1}^{t} \gamma(E^{(k)}, w_{k-1}) \boldsymbol{e}_{w_s} \\
&= \boldsymbol{e}_{w_t} - \boldsymbol{e}_{w_s} = \boldsymbol{0},
\end{aligned}$$

where the last equality holds since $w_t = w_s$. If $\mathrm{nontriv}^{(t)}(w_t) = \mathrm{false}$, we can show that $\prod_{k=s+1}^{t} \gamma(E^{(k)}, w_{k-1}) = 1$. Therefore, we have

$$\begin{aligned}
&\boldsymbol{e}_{w_t} + \left(\prod_{k=s+1}^{t} \gamma(E^{(k)}, w_{k-1}) - 1 \right) A\boldsymbol{r}_{w_s}^{(s)} - \prod_{k=s+1}^{t} \gamma(E^{(k)}, w_{k-1}) \boldsymbol{e}_{w_s} \\
&= \boldsymbol{e}_{w_t} - \boldsymbol{e}_{w_s} = \boldsymbol{0}.
\end{aligned}$$

In either case, we have $A\boldsymbol{r}^* = \boldsymbol{0}$. ◀

Combining Lemma 11 and Lemma 14, we obtain Lemma 10.

Now, we are ready to show Proposition 8.

Proof of Proposition 8. Note that subroutines DualFeasibility, DualSolution, and FarkasCertificateOfDualInfeasibility constitute a certifying algorithm for the feasibility for the dual LP problem (4) (Algorithm 1). The correctness of this algorithm follows from Lemmas 9, 11, and 14.

Now, we analyze the running time of the algorithm. The most time-consuming part of the algorithm is the for-loop from line 2 to 19 in DualFeasibility. This for-loop has m iterations, and $\mathrm{O}(mn)$ operations for computing $\boldsymbol{r}_v^{(k)}$ each $v \in V$ in each iteration. Hence, it takes $\mathrm{O}(m^3 n)$ time. Moreover, since in each of the m iterations the numbers glow $\mathrm{O}(\max(\max_{i,j}(A_{ij}), \max_i(b_i), \max_j(c_j)) \cdot n)$ time, the bit-lengths of the numbers appearing during the algorithm can be bounded by a polynomial in the size of the input. Hence, the algorithm is a strongly polynomial time one. ◀

3.2 A certifying algorithm for the feasibility for the primal LP problem

In this subsection, we provide a certifying algorithm for the feasibility for the primal LP problem (2) with a gainfree Leontief substitution system, using the data computed in DualFeasibility. More precisely, we show that subroutines PrimalFeasibility (Algorithm 6), PrimalSolution (Algorithm 7), and FarkasCertificateOfPrimalInfeasibility (Algorithm 8), together with DualFeasibility, constitute a certifying algorithm for the feasibility for the primal LP problem (2) (Algorithm 5). PrimalFeasibility determines feasibility of the primal LP problem (2) using the same criterion as in (ii) of Theorem 3.6 in [16]. PrimalSolution is similar to PrimalRetrieval in [16]; however, PrimalSolution also computes a primal feasible solution when the dual LP problem is infeasible. FarkasCertificateOfPrimalInfeasibility returns a Farkas' certificate of the primal infeasibility, where the gainfree property is again crucial for the correctness.

The following example shows how these algorithms work.

▶ **Example 18.** Recall Example 7 in Subsection 3.1. In this example, we have $\mathrm{nontriv}^{(m)}(v) = \mathrm{true}$ for each $v \in V$ and $\mathrm{PrimalFeasibility}(\boldsymbol{b}, \mathrm{nontriv}^{(m)}) = \mathrm{true}$ for any $\boldsymbol{b}(\geq \boldsymbol{0})$. Hence, PrimalSolution is called in Algorithm 5. As the dual LP problem is infeasible, $\mathrm{DualFeasibility}(A, \boldsymbol{0})$ is called in PrimalSolution and in particular $\boldsymbol{q} = (1, 1, 1, 1)^T$ is obtained.

Algorithm 5 Combinatorial certifying algorithm for the feasibility for the primal LP problems with gainfree Leontief substitution systems.

Input: A matrix A and vectors $\boldsymbol{b}$ and $\boldsymbol{c}$ for the primal LP problem (2).
$(\boldsymbol{y}^{(m)}, \boldsymbol{r}^{(k)}(k=0,...,m), \text{change}^{(k)}(k=0,...,m), p^{(k)}(k=0,...,m), \text{nontriv}^{(m)}, \boldsymbol{q}, \text{VALUE}) \leftarrow \text{DualFeasibility}(A, \boldsymbol{c})$.
if $\textit{PrimalFeasibility}(\boldsymbol{b}, \text{nontriv}^{(m)}) = \text{true}$ **then**
  $\boldsymbol{x}^* \leftarrow \text{PrimalSolution}(A, \boldsymbol{b}, \text{nontriv}^{(m)}, p^{(k)}(k=0,...,m), \boldsymbol{q}, \text{VALUE})$.
  print "primal-feasible" and **return** $\boldsymbol{x}^*$.
else
  $\boldsymbol{z}^* \leftarrow \text{FarkasCertificateOfPrimalInfeasibility}(\boldsymbol{y}^{(m)}, \text{nontriv}^{(m)})$.
  print "primal-infeasible" and **return** $\boldsymbol{z}^*$.
end

Then in the while-loop in PrimalSolution variables $\boldsymbol{x}^*$ and $\boldsymbol{f}$ are updated as follows. Initially, $\boldsymbol{x}^* = \boldsymbol{0}$ and $\boldsymbol{f} = (b_1, b_2, b_3, b_4)^T$. First, we may choose v_1 according to $\boldsymbol{q}$ and since $p^{(1)}(v_1) = E_4$, $x^*(E_4) = b_1$ and $\boldsymbol{f}$ remains unchanged. Then we may choose v_2 and since $p^{(1)}(v_2) = E_5$, $x^*(E_5) = b_2$ and $\boldsymbol{f}$ remains unchanged. Then we may choose v_3 and since $p^{(1)}(v_3) = E_6$, $x^*(E_6) = b_3$ and $\boldsymbol{f}$ remains unchanged. Finally, we choose v_4 and since $p^{(1)}(v_4) = E_7$, $x^*(E_7) = b_4$. Then we obtain a feasible solution $\boldsymbol{x}^* = (0, 0, 0, b_1, b_2, b_3, b_4)^T$ of the primal LP problem (2).

Algorithm 6 PrimalFeasibility.

Input: A vector $\boldsymbol{b}$ and $\text{nontriv}^{(m)}$.
if $b(v) = 0$ *for all* v *with* $\text{nontriv}^{(m)}(v) = \text{false}$ **then**
  return true.
else
  return false.
end

Now, we show correctness of subroutines PrimalFeasibility, PrimalSolution, and FarkasCertificateOfPrimalInfeasibility, and show the following proposition.

▶ **Proposition 19.** *Algorithm 5 is a combinatorial strongly polynomial time certifying algorithm that runs in* $O(m^3 n)$ *time for the feasibility for the primal LP problem* (2) *with a gainfree Leontief substitution system.*

To show Proposition 19, we use the following lemmas.

▶ **Lemma 20.** *If PrimalFeasibility returns* true*, then the primal LP problem* (2) *is feasible and PrimalSolution returns a feasible solution of* (2)*.*

▶ **Lemma 21.** *If PrimalFeasibility returns* false*, then the primal LP problem* (2) *is infeasible and FarkasCertificateOfPrimalInfeasibility returns a Farkas' certificate of the primal infeasibility.*

Proof of Proposition 19. Note that subroutines PrimalFeasibility, PrimalSolution, and FarkasCertificateOfPrimalInfeasibility, together with DualFeasibility, constitute a certifying algorithm for the feasibility for the primal LP problem (2) (Algorithm 5). Correctness of this algorithm follows from Lemmas 20 and 21.

Algorithm 7 PrimalSolution.

Input: A matrix A and a vector $\boldsymbol{b}$ for the constraint of the primal LP problem (2), and $\mathrm{nontriv}^{(m)}, p^{(k)}(k=0,...,m), \boldsymbol{q}, \mathrm{VALUE}$.

1 **if** VALUE = false **then**
2 $\quad(\boldsymbol{y}^{(m)}, \boldsymbol{r}^{(k)}(k=0,...,m), \mathrm{change}^{(k)}(k=0,...,m), p^{(k)}(k=0,...,m), \boldsymbol{q}, \mathrm{VALUE}) \leftarrow$ DualFeasibility$(A, \mathbf{0})$.
3 **end**
4 For each $E \in \mathcal{E}$, $x^*(E) \leftarrow 0$, $\tilde{V} \leftarrow \{i \in V \mid \mathrm{nontriv}^{(m)}(i) = \mathrm{true}\}$, and for each $v \in V$, $f(v) \leftarrow b(v)$.
5 **while** $\tilde{V} \neq \emptyset$ **do**
6 $\quad$Choose an arbitrary $v \in \tilde{V}$ with maximum $q(v)$.
7 $\quad E \leftarrow p^{(q(v))}(v)$.
8 $\quad x^*(E) \leftarrow f(v)$.
9 $\quad f(u) \leftarrow f(u) + \gamma(E,u)x(E)$ for each $u \in T(E)$.
10 $\quad \tilde{V} \leftarrow \tilde{V} \setminus \{v\}$.
11 **end**
12 **return** $\boldsymbol{x}^*$.

Algorithm 8 FarkasCertificateOfPrimalInfeasibility.

Input: A vector $\boldsymbol{y}^{(m)}$ and $\mathrm{nontriv}^{(m)}$.

1 **for** *each* $v \in V$ **do**
2 $\quad$**if** $\mathrm{nontriv}^{(m)}(v) = \mathrm{true}$ **then**
3 $\quad\quad z^*(v) \leftarrow 0$.
4 $\quad$**else**
5 $\quad\quad z^*(v) \leftarrow$ the coefficient of M in $y^{(m)}(v)$.
6 $\quad$**end**
7 **end**
8 **return** $\boldsymbol{z}^*$.

Now, we analyze the running time of the above algorithm. The most time-consuming part of is DualFeasibility, which runs in $O(m^3 n)$ time as shown in the proof of Proposition 8. This completes the proof. ◀

3.3 Proof of the main theorem (Theorem 6)

Combining the results in Subsections 3.1 and 3.2, we can show our main theorem.

Proof of Theorem 6. From Theorem 4, Algorithms 1 and 5 constitute a certifying algorithm for solving the LP problem. Correctness and the running time of the algorithm follow from Propositions 8 and 19. ◀

Finally, the following example shows that gainfreeness is necessary for convergence of the for-loop from line 2 to 19 in DualFeasibility for a feasible dual LP problem of an LP problem with a Leontief substitution system.

▶ **Example 22.** For the following matrix A (which is Leontief but not gainfree) and vector $\boldsymbol{c}$

$$A = \begin{pmatrix} 1 & -1 \\ -(1/2) & 1 \end{pmatrix} \text{ and } \boldsymbol{c} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

it is clear that $(0,0)^T$ is a feasible solution of the dual LP problem (4). However, one can see that $\boldsymbol{y}^{(0)} = (M,M)^T$, $\boldsymbol{y}^{(1)} = ((1/2)M,M)^T$, $\boldsymbol{y}^{(2)} = ((1/2)M,(1/2)M)^T$, $\boldsymbol{y}^{(3)} = ((1/4)M,(1/2)M)^T$, $\boldsymbol{y}^{(3)} = ((1/4)M,(1/4)M)^T$, and so on, and the for-loop from line 2 to 19 in DualFeasibility does not converge in a finite number of iterations.

4 Conclusion

We proposed a combinatorial strongly polynomial time certifying algorithm for the LP problems with gainfree Leontief substitution systems. Since the dual LP problems with gainfree Leontief substitution systems contains the feasibility for unit-positive Horn systems, we resolved the open questions raised in [15].

An interesting future direction would be to make other non-certifying algorithms certifying. A candidate would be to extend our result on unit Horn systems to unit *q-Horn* systems, introduced in [18]. Unit q-Horn systems include not only unit Horn systems but also unit-two-variable-per-inequality (UTVPI) systems, and the feasibility for unit q-Horn systems is solvable in polynomial time [18]. Furthermore, a certifying algorithm for the feasibility for UTVPI systems is known [25]. Therefore, giving a certifying algorithm for the feasibility for unit q-Horn systems would be an interesting future work.

References

1 Ilan Adler and Steven Cosares. A strongly polynomial algorithm for a special class of linear programs. *Operations Research*, 39:955–960, 1991.

2 Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

3 Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI'22)*, 2022.

4 Riccardo Cambini, Giorgio Gallo, and Maria Grazia Scutellà. Flows on hypergraphs. *Mathematical Programming*, 78(2):195–217, 1997.

5 R. Chandrasekaran and K. Subramani. A combinatorial algorithm for horn programs. *Discrete Optimization*, 10:85–101, 2013.

6 Maria Chudnovsky, Jan Goedgebeur, Oliver Schaudt, and Mingxian Zhong. Obstructions for three-coloring graphs with one forbidden induced subgraph. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1774–1783. SIAM, 2016.

7 Derek G Corneil, Barnaby Dalton, and Michel Habib. Ldfs-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM Journal on Computing*, 42(3):792–807, 2013.

8 Richard W. Cottle and Arthur F. Veinott, Jr. Polyhedral sets having a least element. *Mathematical Programming*, 3:238–249, 1972.

9 George B. Dantzig. Optimal solution of a dynamic leontief model with substitution. *Econometrica*, 23(3):295–302, 1955.

10 Marcel Dhiflaoui, Stefan Funke, Carsten Kwappik, Kurt Mehlhorn, Michael Seel, Elmar Schömer, Ralph Schulte, and Dennis Weber. Certifying and repairing solutions to large lps how good are lp-solvers? In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 255–256, 2003.

11 Lester R Ford Jr. Network flow theory. Technical report, Rand Corp Santa Monica Ca, 1956.

12 Loukas Georgiadis and Robert E Tarjan. Dominator tree certification and divergent spanning trees. *ACM Transactions on Algorithms (TALG)*, 12(1):1–42, 2015.

13 Fred Glover. A bound escalation method for the solution of integer linear programs. *Cahiers du Centre d'Etudes de Recherche Operationelle*, 6(3):131–168, 1964.

14 Andrew V Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995.

15 Pratik Bijaiprakash Gupta. A certifying algorithm for Horn constraint systems. Master's thesis, The University of Texas at Dallas, 2014.

16 Robert G Jeroslow, Kipp Martin, Ronald L Rardin, and Jinchang Wang. Gainfree leontief substitution flow problems. *Mathematical Programming*, 57(1):375–414, 1992.

17 Haim Kaplan and Yahav Nussbaum. Certifying algorithms for recognizing proper circular-arc graphs and unit circular-arc graphs. *Discrete Applied Mathematics*, 157(15):3216–3230, 2009.

18 Kei Kimura and Kazuhisa Makino. Trichotomy for integer linear systems based on their sign patterns. *Discrete Applied Mathematics*, 200:67–78, 2016. doi:10.1016/j.dam.2015.07.004.

19 Kei Kimura and Kazuhisa Makino. A combinatorial certifying algorithm for linear programming problems with gainfree leontief substitution systems. *arXiv*, 2023. arXiv:2306.03368.

20 Dieter Kratsch, Ross M McConnell, Kurt Mehlhorn, and Jeremy P Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. *SIAM Journal on Computing*, 36(2):326–353, 2006.

21 Wassily W Leontief. *The structure of American economy, 1919-1939*. Oxford University Press, second edition, 1951.

22 Ross M McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011.

23 Kurt Mehlhorn, Stefan Naher, and Stefan Näher. *LEDA: A platform for combinatorial and geometric computing*. Cambridge university press, 1999.

24 Kurt Mehlhorn, Adrian Neumann, and Jens M Schmidt. Certifying 3-edge-connectivity. *Algorithmica*, 77(2):309–335, 2017.

25 Antoine Miné. The octagon abstract domain. *Higher-order and symbolic computation*, 19(1):31–100, 2006.

26 Edward F Moore. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959*, pages 285–292, 1959.

27 Jens M Schmidt. Contractions, removals, and certifying 3-connectivity in linear time. *SIAM Journal on Computing*, 42(2):494–535, 2013.

28 Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

29 K Subramani and Piotr Wojciechowski. A combinatorial certifying algorithm for linear feasibility in utvpi constraints. *Algorithmica*, 78(1):166–208, 2017.

30 K Subramani and James Worthington. A new algorithm for linear and integer feasibility in horn constraints. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 215–229. Springer, 2011.

31 J.D. Ullman and A. Van Gelder. Efficient test for top-down termination of logical rules. *Journal of the Association for Computing Machinery*, 35:345–373, 1988.

32 Hans Van Maaren and Chuangyin Dang. Simplicial pivoting algorithms for a tractable class of integer programs. *Journal of Combinatorial Optimization*, 6(2):133–142, 2002.

Reconfiguration of the Union of Arborescences

Yusuke Kobayashi
Research Institute for Mathematical Sciences, Kyoto University, Japan

Ryoga Mahara
Department of Mathematical Informatics, University of Tokyo, Japan

Tamás Schwarcz
MTA-ELTE Momentum Matroid Optimization Research Group, Department of Operations Research, ELTE Eötvös Loránd University, Budapest, Hungary

Abstract

An arborescence in a digraph is an acyclic arc subset in which every vertex except a root has exactly one incoming arc. In this paper, we show the reconfigurability of the union of k arborescences for fixed k in the following sense: for any pair of arc subsets that can be partitioned into k arborescences, one can be transformed into the other by exchanging arcs one by one so that every intermediate arc subset can also be partitioned into k arborescences. This generalizes the result by Ito et al. (2023), who showed the case with $k = 1$. Since the union of k arborescences can be represented as a common matroid basis of two matroids, our result gives a new non-trivial example of matroid pairs for which two common bases are always reconfigurable to each other.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Mathematics of computing → Combinatorial optimization

Keywords and phrases Arborescence packing, common matroid basis, combinatorial reconfiguration

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.48

Related Version *Full Version*: `https://arxiv.org/abs/2304.13217`

Funding This work was supported by the Research Institute for Mathematical Sciences, an International Joint Usage/Research Center located in Kyoto University and by the Lendület Programme of the Hungarian Academy of Sciences – grant number LP2021-1/2021.
Yusuke Kobayashi: Yusuke Kobayashi was supported by JSPS KAKENHI Grant Numbers JP20H05795, JP20K11692, and JP22H05001.
Tamás Schwarcz: Tamás Schwarcz was supported by the ÚNKP-22-3 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

Acknowledgements The authors thank members of the project "Fusion of Computer Science, Engineering and Mathematics Approaches for Expanding Combinatorial Reconfiguration" for discussion on this topic. The authors are grateful to András Frank for bringing the paper [1] to their attention.

1 Introduction

1.1 Reconfigurability of Common Bases of Matroids

Exchanging a pair of elements, i.e., adding one element to a set and removing another element from it, is a fundamental operation in matroid theory. The basis exchange axiom for matroids implies that, for any pair of bases of a matroid, one can be transformed into the other by repeatedly exchanging pairs of elements so that all the intermediate sets are also bases. That is, the basis family of a matroid is connected with respect to element exchanges. This is an important property of matroid basis families that is used in various contexts, e.g., it is a key to show the validity of a local search algorithm for finding a maximum weight basis.

In contrast to matroid basis families, a family of common bases of two matroids does not necessarily enjoy this property. More precisely, for two matroids M_1 and M_2 over a common ground set, the following condition, which we call *Reconfigurability of Common Bases (RCB)*, does not necessarily hold.

(RCB) For any pair of common bases B and B' of two matroids M_1 and M_2, there exists a sequence of common bases $B_0, B_1, \ldots, B_\ell$ such that $B_0 = B$, $B_\ell = B'$, B_i is a common basis, and $|B_{i-1} \setminus B_i| = |B_i \setminus B_{i-1}| = 1$ for each $i \in \{1, 2, \ldots, \ell\}$.

As an example, suppose that G is a cycle of length four, which has exactly two perfect matchings. Since G is a bipartite graph, the family of all the perfect matchings in G can be represented as a family of common bases of two matroids, and we see that (RCB) does not hold in this setting.

On the other hand, there are some special cases satisfying (RCB). When M_1 is a graphic matroid and M_2 is the dual matroid of M_1, it is known that (RCB) holds [10]. A conjecture by White [28] that (RCB) holds when M_1 is an arbitrary matroid and M_2 is its dual has been open for more than 40 years. The conjecture was verified for strongly base orderable matroids [21] and for sparse paving matroids [4]. Recently, the conjecture was settled for split matroids [3], a large class containing paving matroids as well.

Another special case with property (RCB) is the family of all arborescences in a digraph. For a digraph $D = (V, A)$ with a specified vertex $r \in V$ called a *root*, an *r-arborescence* is an acyclic arc subset of A in which every vertex in $V \setminus \{r\}$ has exactly one incoming arc. When the root vertex is not specified, it is simply called an *arborescence*. We see that an r-arborescence (or an arborescence) is represented as a common basis of the graphic matroid corresponding to the acyclic constraint and the partition matroid corresponding to the indegree constraint. It is shown by Ito et al. [17] that the family of all arborescences (or r-arborescences) in a digraph satisfies (RCB).

1.2 Our Results

In this paper, we study the union of k arc-disjoint r-arborescences, where k is a fixed positive integer. For a digraph $D = (V, A)$ and a root $r \in V$, let $\mathcal{F}_{k,r} \subseteq 2^A$ denote the set of all arc subsets that can be partitioned into k arc-disjoint r-arborescences. It is known that $\mathcal{F}_{k,r}$ is represented as the common bases of two matroids M_1 and M_2, where M_1 is the union of k graphic matroids and M_2 is the direct sum of uniform matroids corresponding to the indegree constraint; see [26, Corollary 53.1c]. The main contribution of this paper is to show that such a pair of M_1 and M_2 satisfies property (RCB). Formally, our main result is stated as follows.

▶ **Theorem 1.** *Let $D = (V, A)$ be a digraph with a root $r \in V$ and let k be a positive integer. Let $\mathcal{F}_{k,r} \subseteq 2^A$ denote the family of all arc subsets that can be partitioned into k arc-disjoint r-arborescences. For any $S, T \in \mathcal{F}_{k,r}$, there exists a sequence $T_0, T_1, \ldots, T_\ell$ such that $T_0 = S$, $T_\ell = T$, $T_i \in \mathcal{F}_{k,r}$ for $i \in \{0, 1, \ldots, \ell\}$, and $|T_{i-1} \setminus T_i| = |T_i \setminus T_{i-1}| = 1$ for $i \in \{1, 2, \ldots, \ell\}$. Furthermore, such a sequence can be found in polynomial time.*

We also prove the analogue of Theorem 1 in the case when a feasible arc set is the union of k arc-disjoint arborescences that may have distinct roots. Formally, our result is stated as follows and will be discussed in Section 4.

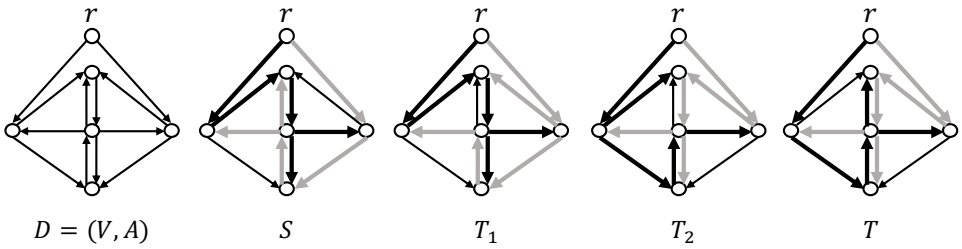


Figure 1 The leftmost figure represents a digraph $D = (V, A)$ with a root $r \in V$. In each figure, the union of thick black and gray arc subsets is in $\mathcal{F}_{k,r}$, where $k = 2$. The sequence $T_0 = S, T_1, T_2, T_3 = T$ is a shortest reconfiguration sequence from S to T.

▶ **Theorem 2.** *Let $D = (V, A)$ be a digraph and let k be a positive integer. Let $\mathcal{F}_k \subseteq 2^A$ denote the family of all arc subsets that can be partitioned into k arc-disjoint arborescences. For any $S, T \in \mathcal{F}_k$, there exists a sequence $T_0, T_1, \ldots, T_\ell$ such that $T_0 = S$, $T_\ell = T$, $T_i \in \mathcal{F}_k$ for $i \in \{0, 1, \ldots, \ell\}$, and $|T_{i-1} \setminus T_i| = |T_i \setminus T_{i-1}| = 1$ for $i \in \{1, 2, \ldots, \ell\}$. Furthermore, such a sequence can be found in polynomial time.*

When $k = 1$, Theorem 1 amounts to the reconfigurability of r-arborescences, which is an easy result; see e.g. [1, 17]. In this special case, we can update an r-arborescence S so that $|S \setminus T|$ decreases monotonically, which immediately leads to the existence of a reconfiguration sequence of length $|S \setminus T|$. Here, the *length* of a reconfiguration sequence is defined as the number of exchange operations in it. Meanwhile, such an update of S is not always possible when $k \geq 2$, that is, there exists an example in which more than $|S \setminus T|$ steps are required to transform S into T; see Example 3. This suggests that the case with $k \geq 2$ is much more complicated than the case with $k = 1$.

▶ **Example 3.** Let $k = 2$, let $D = (V, A)$ be a digraph with a root $r \in V$, and let $S, T \in \mathcal{F}_{k,r}$ be as in Figure 1. Then, the shortest reconfiguration sequence between S and T has length 3, while $|S \setminus T| = |T \setminus S| = 2$.

We here give another remark on the relationship between the case of $k \geq 2$ and the case of $k = 1$. We have already mentioned that an r-arborescence is represented as a common basis of the graphic matroid M_1 and the partition matroid M_2 corresponding to the indegree constraint. Then, $\mathcal{F}_{k,r}$ is represented as a family of common bases of kM_1 and kM_2, where kM_i is the matroid whose bases are the unions of k disjoint bases of M_i. Since the matroids representing $\mathcal{F}_{k,r}$ have these special forms, to prove Theorem 1, it will be natural to expect that the case of $k \geq 2$ is reduced to the case of $k = 1$. However, the following theorem suggests that such an approach does not work naively; see Section 5 for the proof.

▶ **Theorem 4.** *There exist matroids $M_1 = (E, \mathcal{B}_1)$ and $M_2 = (E, \mathcal{B}_2)$ such that the pair of matroids (M_1, M_2) satisfies (RCB), while $(2M_1, 2M_2)$ does not satisfy it.*

1.3 Related Work

The study of packing arborescences was initiated by Edmonds [9], who showed that a digraph $D = (V, A)$ contains k arc-disjoint r-arborescences if and only if every non-empty subset of $V \setminus \{r\}$ has at least k entering arcs; see Theorem 5. Lovász [22] gave a simpler proof for this theorem. There are several directions of extension of Edmonds' theorem. The first one by Frank [12] is to extend directed graphs to mixed graphs. Second, Frank, Király, and

Király [13] extended directed graphs to directed hypergraphs. Third, Kamiyama, Katoh, and Takizawa [19] and Fujishige [14] extended the problem to the packing of rooted-trees that cover only reachable vertex set. Lastly, de Gevigney, Nguyen, and Szigeti [8] considered the packing of rooted-trees with matroid constraints. By combining these extensions, we can consider further generalizations, which have been actively studied in [2, 11, 15, 20, 23].

Combinatorial reconfiguration is an emerging field in discrete mathematics and theoretical computer science. One of the central problems in combinatorial reconfiguration is the following algorithmic question; for two given discrete structures, determine whether one can be transformed into the other by a sequence of local changes. See surveys of Nishimura [24] and van den Heuvel [27], and see also solvers for combinatorial reconfiguration [18].

In this framework, if we focus on common bases of two matroids, then we can consider the following reconfiguration problem.

▶ **Matroid Intersection Reconfiguration.**

Input. Two matroids M_1 and M_2 and their common bases B and B'.

Question. Determine whether B can be transformed into B' by exchanging a pair of elements repeatedly so that all the intermediate sets are common bases of M_1 and M_2.

Although this problem seems to be a fundamental problem, its polynomial solvability is still open. If the pair of input matroids M_1 and M_2 satisfies (RCB), then Matroid Intersection Reconfiguration is trivial, i.e., the transformation is always possible. If each common basis corresponds to a maximum matching in a bipartite graph, then the pair of M_1 and M_2 does not necessarily satisfy (RCB), but Matroid Intersection Reconfiguration can be solved in polynomial time [16]. Actually, it is shown in [16] that the reconfiguration problem of maximum matchings in non-bipartite graphs is also solvable in polynomial time.

Since the spanning trees in a graph form a matroid basis family, the reconfiguration problem on spanning trees is trivial, i.e., the transformation is always possible. However, the problem becomes non-trivial if we add some constraints. Reconfiguration problems on spanning trees with additional constraints were studied in [5, 6].

1.4 Overview

We now describe an outline of the proof of Theorem 1. For feasible arc subsets S and T, in order to show that S can be transformed into T, it suffices to show that S can be transformed into another feasible arc subset S' such that $|S' \setminus T| = |S \setminus T| - 1$. When $k = 1$, such S' can be easily obtained from S by exchanging only one pair of arcs; see [1, 17]. However, this is not indeed the case when $k \geq 2$ as shown in Example 3, that is, several steps may be required to obtain S'. This is the main technical difficulty of the problem.

To overcome this difficulty, we introduce and use *minimal tight sets* and an *auxiliary digraph*. Let $T \setminus S = \{f_1, \ldots, f_p\}$. For each arc $f_i \in T \setminus S$, we consider the inclusionwise minimal vertex set X_i subject to X_i contains f_i and exactly k arcs in S enter X_i (i.e., X_i is *tight*). Then, X_i gives a characterization of arcs $e \in S$ that can be replaced with f_i; see Lemma 9. By using $X_1, \ldots, X_p$, we construct an auxiliary digraph H, whose definition is given in Section 3.2. Roughly speaking, H is similar to the exchangeability digraph when $\mathcal{F}_{k,r}$ is represented as the common bases of two matroids. Then, we can show that H has a dicycle; see Lemma 12. If H has a self-loop, then we can obtain a desired arc subset S' by exchanging only one pair of arcs. Otherwise, we update the arc set S so that the length of the shortest dicycle in H becomes shorter, which is discussed in Section 3.3. By repeating this procedure, we obtain a desired arc subset S' in finite steps.

The rest of the paper is organized as follows. In Section 2, we introduce some notation and show basic results on arborescence packing. In Section 3, we give a proof of Theorem 1,

which is the main part of the paper. In Section 4, we show the analogue of Theorem 1 in the case when arborescences may have distinct roots and give a proof of Theorem 2. In Section 5, we show that (RCB) is not closed under sums (Theorem 4) by giving a concrete example. Finally, in Section 6, we conclude this paper by giving some remarks.

2 Preliminaries

Let $D = (V, A)$ be a digraph that may have parallel arcs. For an arc $e \in A$, the head and tail of e are denoted by $\mathrm{head}(e)$ and $\mathrm{tail}(e)$, respectively. For $e \in A$ and $X \subseteq V$, we say that X *contains* e if X contains both $\mathrm{head}(e)$ and $\mathrm{tail}(e)$. For $F \subseteq A$ and $X, Y \subseteq V$, let $\Delta_F(X, Y)$ denote the set of arcs in F from X to Y, i.e., $\Delta_F(X, Y) = \{e \in F \mid \mathrm{tail}(e) \in X,\ \mathrm{head}(e) \in Y\}$. Let $\Delta_F^-(X)$ denote $\Delta_F(V \setminus X, X)$. Let $\delta_F(X, Y) = |\Delta_F(X, Y)|$ and $\delta_F^-(X) = |\Delta_F^-(X)|$. For $v \in V$, $\delta_F^-(\{v\})$ is simply denoted by $\delta_F^-(v)$.

For a digraph $D = (V, A)$ with a specified vertex $r \in V$ called a *root*, an *r-arborescence* is an acyclic arc subset $T \subseteq A$ such that $\delta_T^-(v) = 1$ for $v \in V \setminus \{r\}$ and $\delta_T^-(r) = 0$. Note that an arborescence is often defined as a subgraph of D in the literature, but it is regarded as an arc subset in this paper. For a positive integer k, let $\mathcal{F}_{k,r} \subseteq 2^A$ denote the family of all arc subsets that can be partitioned into k arc-disjoint r-arborescences. If r and k are clear, an arc subset in $\mathcal{F}_{k,r}$ is simply called *feasible*. Note that for any feasible arc subsets S and T, $|S| = |T|$ holds. Edmonds [9] gave the following characterization of feasible arc subsets.

▶ **Theorem 5** (Edmonds [9]). *For a digraph $D = (V, A)$ with $r \in V$, an arc subset $T \subseteq A$ is in $\mathcal{F}_{k,r}$ if and only if $\delta_T^-(v) = k$ for any $v \in V \setminus \{r\}$, $\delta_T^-(r) = 0$, and $\delta_T^-(X) \geq k$ for any $X \subseteq V \setminus \{r\}$ with $X \neq \emptyset$.*

For a feasible arc subset $T \subseteq A$, we say that a vertex set $X \subseteq V \setminus \{r\}$ is *tight with respect to* T if $\delta_T^-(X) = k$. It is well-known that the tight sets are closed under intersection and union as follows.

▶ **Lemma 6.** *Let $T \subseteq A$ be a feasible arc set and let $X, Y \subseteq V \setminus \{r\}$ be tight sets with respect to T with $X \cap Y \neq \emptyset$. Then, $X \cap Y$ and $X \cup Y$ are tight sets with respect to T. Furthermore, T has no arc connecting $X \setminus Y$ and $Y \setminus X$.*

Proof. By a simple counting argument, we obtain

$$\begin{aligned} k + k &= \delta_T^-(X) + \delta_T^-(Y) \\ &= \delta_T^-(X \cap Y) + \delta_T^-(X \cup Y) + \delta_T(X \setminus Y, Y \setminus X) + \delta_T(Y \setminus X, X \setminus Y) \\ &\geq k + k + 0 + 0, \end{aligned}$$

which shows that $\delta_T^-(X \cap Y) = \delta_T^-(X \cup Y) = k$ and $\delta_T(X \setminus Y, Y \setminus X) = \delta_T(Y \setminus X, X \setminus Y) = 0$. This means that $X \cap Y$ and $X \cup Y$ are tight and T has no arc connecting $X \setminus Y$ and $Y \setminus X$. ◀

For a positive integer p, let $[p] = \{1, 2, \ldots, p\}$. For feasible arc subsets $S, T \in \mathcal{F}_{k,r}$, we say that $T_0, T_1, \ldots, T_\ell$ is a *reconfiguration sequence between S and T* if $T_0 = S$, $T_\ell = T$, $T_i \in \mathcal{F}_{k,r}$ for any $i \in [\ell] \cup \{0\}$, and $|T_{i-1} \setminus T_i| = |T_i \setminus T_{i-1}| = 1$ for any $i \in [\ell]$. We call ℓ the *length* of the sequence. With this terminology, Theorem 1 is rephrased as follows: for any $S, T \in \mathcal{F}_{k,r}$, there always exists a reconfiguration sequence between S and T.

We denote a matroid on ground set E with family of bases $\mathcal{B}$ by $M = (E, \mathcal{B})$. See [25] for the definition and basic properties of matroids.

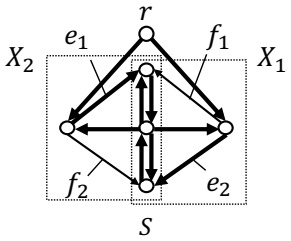


Figure 2 Minimal tight sets X_i.

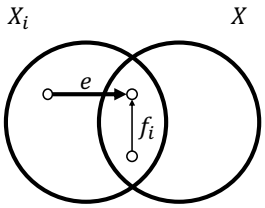


Figure 3 Proof of Lemma 9.

3 Proof of Theorem 1

In this section, we prove Theorem 1. For a digraph $D = (V, A)$ with a root r, let $S, T \in \mathcal{F}_{k,r}$ be feasible arc subsets. Let $S \setminus T = \{e_1, \dots, e_p\}$ and $T \setminus S = \{f_1, \dots, f_p\}$, where $p = |S \setminus T|$. By changing the indices if necessary, we may assume that $\mathrm{head}(e_i) = \mathrm{head}(f_i)$ for any $i \in [p]$. To prove Theorem 1, it suffices to show that we can transform S to a new feasible arc subset S' such that $|S' \setminus T| = p - 1$, which is formally stated as follows.

▶ **Proposition 7.** *Let $S \subseteq A$ and $T \subseteq A$ be feasible arc subsets with $|S \setminus T| = p$. Then, there is a new feasible arc subset $S' \subseteq A$ such that $|S' \setminus T| = p - 1$ and there is a reconfiguration sequence between S and S'.*

In what follows in this section, we give a proof of Proposition 7 and prove Theorem 1. In Section 3.1, we introduce a *minimal tight set* X_i for each $i \in [p]$ and show some properties of X_i. In Section 3.2, we construct an *auxiliary digraph* using X_i and show its properties. In Section 3.3, we show that S can be modified so that the shortest dicycle length in the auxiliary digraph becomes shorter until the desired S' is found. Finally, we prove Proposition 7 and Theorem 1 in Section 3.4.

3.1 Minimal Tight Sets

For $i \in [p]$, define $X_i \subseteq V$ as the inclusionwise minimal tight vertex set with respect to S that contains f_i. For notational convenience, define $X_i = V$ if no such tight set exists. Note that such X_i is uniquely defined since tight sets are closed under intersection; see Lemma 6.

▶ **Example 8.** Let k, $D = (V, A)$, S, and T be as in Example 3. Then, X_1 and X_2 are as shown in Figure 2.

We show some properties of X_i.

▶ **Lemma 9.** *Let $i \in [p]$. If $e \in S$ satisfies $\mathrm{head}(e) = \mathrm{head}(f_i)$ and e is contained in X_i, then $S' = S - e + f_i$ is feasible.*

Proof. Since $\delta^-_{S'}(v) = k$ for any $v \in V \setminus \{r\}$ and $\delta^-_{S'}(r) = 0$, by Theorem 5, it suffices to show that $\delta^-_{S'}(X) \geq k$ holds for any $X \subseteq V \setminus \{r\}$ with $X \neq \emptyset$.

Assume to the contrary that there exists a nonempty subset X of $V \setminus \{r\}$ with $\delta^-_{S'}(X) < k$. Since $\delta^-_S(X) \geq k$, we obtain $\delta^-_S(X) = k$, $e \in \Delta^-_S(X)$, and $f_i \notin \Delta^-_A(X)$. Hence, $\mathrm{head}(e) = \mathrm{head}(f_i) \in X$, $\mathrm{tail}(e) \notin X$, and $\mathrm{tail}(f_i) \in X$ (Figure 3). This shows that X is a tight set with respect to S containing f_i. By Lemma 6, $Y := X \cap X_i$ is also a tight set with respect to S containing f_i, which contradicts the minimality of X_i as $\mathrm{tail}(e) \in X_i \setminus Y$. ◀

Suppose that $f_1' \in S \cap T$ is an arc such that $\mathrm{head}(f_1') = \mathrm{head}(f_1)$ and f_1' is contained in X_1. By Lemma 9, $S' := S - f_1' + f_1$ is feasible. For each $i \in [p]$, define $X_i' \subseteq V$ as the counterpart of X_i associated with S', i.e., define X_i' as the inclusionwise minimal tight vertex set with respect to S' that contains f_i (f_i' if $i = 1$), and $X_i' = V$ if no such set exists.

▶ **Lemma 10.** *Let f_1' and X_1' be as above. Then, it holds that $X_1' = X_1$.*

Proof. We first show $X_1' \subseteq X_1$. If $X_1 = V$, then $X_1' \subseteq V = X_1$ is obvious. Otherwise, since X_1 is a tight set with respect to S that contains both f_1 and f_1', we obtain $\delta^-_{S'}(X_1) = \delta^-_S(X_1) = k$, that is, X_1 is a tight set with respect to S'. This shows that $X_1' \subseteq X_1$ by the minimality of X_1'.

We next show $X_1' \supseteq X_1$. If $X_1' = V$, then $X_1' = V \supseteq X_1$ is obvious. Otherwise, since X_1' is a tight set with respect to S' that contains f_1', we obtain $k = \delta^-_{S'}(X_1') \geq \delta^-_S(X_1') \geq k$, which shows that X_1' is a tight set with respect to S. This shows that $X_1' \supseteq X_1$ by the minimality of X_1. This completes the proof. ◀

Note that Lemma 10 shows that the roles of S and S' are symmetric by replacing f_1 with f_1'. The following lemma shows a relationship between X_i and X_i', which plays a key role in our argument.

▶ **Lemma 11.** *Let f_1' be an arc and X_i' be the vertex set for $i \in [p]$ as above. Let $e \in S$ be an arc contained in X_1. For $i \in [p]$, we have one of the following:*

1. *$X_i = X_i'$,*
2. *X_i contains e, or*
3. *X_i' contains e.*

Proof. By Lemma 10, it suffices to consider the case when $i \in [p] \setminus \{1\}$. If $X_i = V$ or $X_i' = V$, then the second or third condition holds, and so we may assume that $X_i, X_i' \subseteq V \setminus \{r\}$.

Assume that $X_i \neq X_i'$. Since the roles of S and S' are symmetric as we have seen in Lemma 10, without loss of generality, we may assume that $X_i \setminus X_i' \neq \emptyset$. Since X_i is the inclusionwise minimal tight set containing f_i with respect to S, $X_i \cap X_i'$ is not a tight set with respect to S, i.e.,

$$\delta^-_S(X_i \cap X_i') \geq k + 1, \tag{1}$$

where we note that $X_i \cap X_i' \neq \emptyset$ as both X_i and X_i' contain f_i; see Figure 4 (left). We also see that

$$\delta^-_S(X_i') \leq \delta^-_{S'}(X_i') + 1 = k + 1, \tag{2}$$

because $\delta^-_{S'}(X_i') = k$ and $S' = S - f_1' + f_1$.

By (1) and (2), we obtain

$$\begin{aligned} k + (k+1) &\geq \delta^-_S(X_i) + \delta^-_S(X_i') \\ &= \delta^-_S(X_i \cap X_i') + \delta^-_S(X_i \cup X_i') + \delta_S(X_i \setminus X_i', X_i' \setminus X_i) + \delta_S(X_i' \setminus X_i, X_i \setminus X_i') \\ &\geq (k+1) + k + 0 + 0. \end{aligned}$$

This shows that all the inequalities are tight, which yields the following:

- **(a)** $\delta^-_S(X_i') = \delta^-_{S'}(X_i') + 1 = k+1$, which implies that $\mathrm{head}(f_1') = \mathrm{head}(f_1) \in X_i'$, $\mathrm{tail}(f_1') \notin X_i'$, and $\mathrm{tail}(f_1) \in X_i'$; see Figure 4 (right),
- **(b)** $X_i \cup X_i'$ is a tight set with respect to S, and
- **(c)** S contains no arc connecting $X_i \setminus X_i'$ and $X_i' \setminus X_i$.

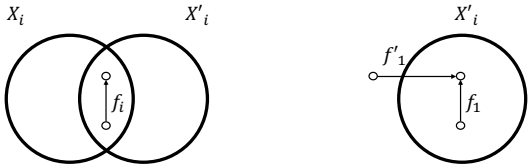


Figure 4 Location of the arcs f_i, f_1, and f'_1.

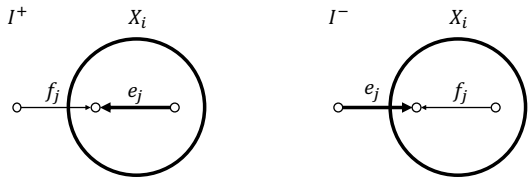


Figure 5 Definitions of I^+ and I^-.

By (a) and (b), $X_i \cup X'_i$ is a tight set with respect to S that contains f_1, and hence $X_1 \subseteq X_i \cup X'_i$, because X_1 is the unique minimal tight set containing f_1. Therefore, any arc $e \in S$ contained in X_1 is also contained in $X_i \cup X'_i$. This together with (c) shows that such e is contained in either X_i or X'_i, which completes the proof. ◀

3.2 Auxiliary Digraph

For two feasible arc subsets S and T, we construct an associated *auxiliary digraph* $H = (V_H, A_H)$ such that $V_H = [p]$ and A_H contains an arc (i, j) if X_i contains e_j. Recall that $X_i \subseteq V$ is the inclusionwise minimal tight vertex set with respect to S that contains f_i, or $X_i = V$ if no such tight set exists. Note that H may contain self-loops. For example, in the case of Example 3 (see also Example 8), H forms a dicycle of length 2. In this subsection, we show some properties of H.

▶ **Lemma 12.** *Every vertex in H has at least one outgoing arc (possibly, a self-loop).*

Proof. Assume to the contrary that $i \in V_H = [p]$ has no outgoing arc in H. Then, by the definition of H, X_i does not contain e_j for any $j \in [p]$. Define $I^+, I^- \subseteq [p]$ as

$$I^+ = \{j \in [p] \mid \mathrm{head}(e_j) = \mathrm{head}(f_j) \in X_i,\ \mathrm{tail}(e_j) \in X_i,\ \mathrm{tail}(f_j) \notin X_i\},$$
$$I^- = \{j \in [p] \mid \mathrm{head}(e_j) = \mathrm{head}(f_j) \in X_i,\ \mathrm{tail}(e_j) \notin X_i,\ \mathrm{tail}(f_j) \in X_i\};$$

see Figure 5. Since X_i contains f_i but does not contain e_i, it holds that $i \in I^-$. We also see that $I^+ = \emptyset$ as X_i does not contain e_j for each j. Then, we obtain

$$k \leq \delta^-_T(X_i) = \delta^-_S(X_i) + |I^+| - |I^-| = k + |I^+| - |I^-| \leq k - 1,$$

which is a contradiction. ◀

▶ **Lemma 13.** *If H has an arc (i, j), then $X_i \cap X_j \neq \emptyset$.*

Proof. Since X_i contains e_j and X_j contains f_j, both X_i and X_j contain the vertex $\mathrm{head}(e_j) = \mathrm{head}(f_j)$. This shows that $X_i \cap X_j \neq \emptyset$. ◀

▶ **Lemma 14.** *If H has a dipath P such that $\bigcup_{i \in V(P)} X_i$ contains some arc $e \in S$, then there exists $i \in V(P)$ such that X_i contains e, where $V(P)$ denotes the set of vertices in P.*

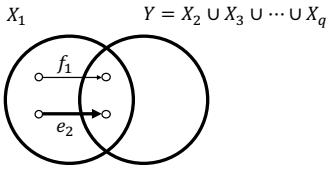


Figure 6 Location of e_2 and f_1.

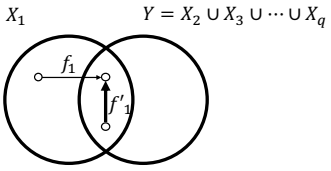


Figure 7 Location of f'_1.

Proof. If $X_i = V$ for some $i \in V(P)$, then the claim is obvious. Thus, it suffices to consider the case when $X_i \neq V$ for any $i \in V(P)$. By renaming the indices if necessary, we may assume that P traverses $1, 2, \ldots, |V(P)|$ in this order. Assume to the contrary that e is not contained in X_i for any $i \in V(P)$. Let $1 \leq i < j \leq |V(P)|$ be indices that minimize $j - i$ subject to $X_i \cup X_{i+1} \cup \cdots \cup X_j$ contains e. Let $Y := X_i \cup X_{i+1} \cup \cdots \cup X_{j-1}$. By applying Lemmas 6 and 13 repeatedly, we see that Y is tight and $Y \cap X_j \neq \emptyset$. Then, Lemma 6 shows that no arc in S connects $Y \setminus X_j$ and $X_j \setminus Y$. Hence, $e \in S$ has to be contained in Y or X_j, which contradicts the minimality of $j - i$. ◀

3.3 Shortest Dicycle

We see that H has a dicycle by Lemma 12. Let C be a shortest dicycle in H, and let q denote its length. If $q = 1$, i.e., H contains a self-loop incident to $i \in V_H$, then Lemma 9 shows that $S' := S - e_i + f_i$ is feasible and $|S' \setminus T| = p - 1$. Thus, in what follows, we consider the case when $q \geq 2$. This implies that $X_i \neq V$ for any $i \in [p]$. By renaming the indices if necessary, we may assume that C traverses $1, 2, \ldots, q \in V_H$ in this order. Let $Y := X_2 \cup X_3 \cup \cdots \cup X_q$. Note that both Y and $X_1 \cap Y$ are tight with respect to S by Lemmas 6 and 13.

▶ **Lemma 15.** *Arc e_2 is not contained in Y.*

Proof. Assume to the contrary that e_2 is contained in Y. Then, by Lemma 14, there exists $i \in \{2, 3, \ldots, q\}$ such that X_i contains e_2. In such a case, since H contains an arc $(i, 2)$ by definition, H has a dicycle traversing $2, 3, \ldots, i$ in this order, which contradicts the choice of C. ◀

▶ **Lemma 16.** *Arc e_2 is from $X_1 \setminus Y$ to $X_1 \cap Y$.*

Proof. Since $\mathrm{head}(e_2) = \mathrm{head}(f_2) \in X_2 \subseteq Y$, Lemma 15 shows that $\mathrm{tail}(e_2) \notin Y$. We also see that e_2 is contained in X_1 as H contains arc $(1, 2)$. By combining them, e_2 is from $X_1 \setminus Y$ to $X_1 \cap Y$. ◀

▶ **Lemma 17.** *Arc f_1 is from $X_1 \setminus Y$ to $X_1 \cap Y$.*

Proof. By definition, f_1 is contained in X_1, which means that $\mathrm{head}(f_1) \in X_1$ and $\mathrm{tail}(f_1) \in X_1$. Furthermore, since e_1 is contained in X_q as H contains arc $(q, 1)$, we have that $\mathrm{head}(f_1) = \mathrm{head}(e_1) \in X_q \subseteq Y$. Thus, it suffices to show that $\mathrm{tail}(f_1) \notin Y$.

Assume to the contrary that $\mathrm{tail}(f_1) \in Y$. Then, $X_1 \cap Y$ contains f_1. Since $X_1 \cap Y$ is a tight set with respect to S by Lemma 6 and $X_1 \cap Y \subseteq X_1 \setminus \{\mathrm{tail}(e_2)\}$ by Lemma 16, this contradicts the minimality of X_1. ◀

See Figure 6 for the illustration of Lemmas 15–17.

▶ **Lemma 18.** *There exists an arc $f'_1 \in S$ with $\mathrm{head}(f'_1) = \mathrm{head}(f_1)$ such that either*

1. *$f'_1 \in S \setminus T$ and f'_1 is contained in X_1, or*
2. *$f'_1 \in S \cap T$ and f'_1 is contained in $X_1 \cap Y$.*

Proof. If $\mathrm{head}(e_2) = \mathrm{head}(f_1)$, then $f_1' := e_2$ satisfies the first condition. Thus, suppose that $\mathrm{head}(e_2) \neq \mathrm{head}(f_1)$. Then, since $\delta_S^-(X_1 \cap Y) = k$, $\delta_S^-(\mathrm{head}(f_1)) = k$, and $e_2 \in \Delta_S^-(X_1 \cap Y) \setminus \Delta_S^-(\mathrm{head}(f_1))$ by Lemma 16, we obtain $\Delta_S^-(\mathrm{head}(f_1)) \setminus \Delta_S^-(X_1 \cap Y) \neq \emptyset$. Therefore, S has an arc f_1' with $\mathrm{head}(f_1') = \mathrm{head}(f_1)$ such that $f_1' \notin \Delta_S^-(X_1 \cap Y)$, which implies that f_1' is contained in $X_1 \cap Y$ (Figure 7). Such an arc f_1' satisfies one of the conditions. ◀

Let f_1' be an arc as in Lemma 18 and let $S' := S - f_1' + f_1$, which is feasible by Lemma 9. If f_1' satisfies the first condition in the lemma (i.e., $f_1' \in S \setminus T$), then $|S' \setminus T| = p - 1$, and hence we are done. Thus, in what follows, we consider the case when f_1' satisfies the second condition in the lemma. In this case, define $X_i' \subseteq V$ for each $i \in [p]$ as in Section 3.1. Define the auxiliary digraph H' associated with S' and T in the same way as H.

▶ **Lemma 19.** *Suppose that f_1' satisfies the second condition in Lemma 18. Then, the auxiliary digraph H' associated with $S' = S - f_1' + f_1$ and T has a dicycle of length at most $q - 1$.*

Proof. We first show that $X_i \neq X_i'$ for some $i \in [q]$. Assume to the contrary that $X_i = X_i'$ for each $i \in [q]$. Then, we see that $Y = X_2 \cup \cdots \cup X_q$ is a tight set with respect to S, and we also see that $Y = X_2' \cup \cdots \cup X_q'$ is tight with respect to S'. This shows that $\delta_S^-(Y) = k = \delta_{S'}^-(Y)$. However, we obtain $\Delta_{S'}^-(Y) = \Delta_S^-(Y) \cup \{f_1\}$ by Lemma 17 and by the second condition in Lemma 18, which is a contradiction.

Therefore, $X_i \neq X_i'$ for some $i \in [q]$. Let i be the minimal index with $X_i \neq X_i'$, where we note that $i \geq 2$ by Lemma 10. Since C is a shortest dicycle, H does not contain an arc $(i, 2)$, that is, X_i does not contain e_2. As $X_i \neq X_i'$ and X_i does not contain e_2, by applying Lemma 11 with $e = e_2$, we see that X_i' contains e_2, which means that H' contains an arc $(i, 2)$. By the minimality of i, $X_j' = X_j$ holds for $j \in \{2, \ldots, i-1\}$, and hence H' contains a dicycle C' traversing $2, 3, \ldots, i$ in this order. Since the length of C' is at most $q - 1$, this completes the proof. ◀

3.4 Putting Them Together

By the above lemmas, we obtain Proposition 7 as follows. Suppose that $S \subseteq A$ and $T \subseteq A$ are feasible arc subsets and H is the auxiliary digraph associated with S and T. If H has a self-loop incident to $i \in V_H$, then $S' := S - e_i + f_i$ satisfies the conditions in Proposition 7. Otherwise, let q be the length of a shortest dicycle in H and let $f_1' \in S$ be an arc satisfying the condition in Lemma 18. By the description just after Lemma 18 and by Lemma 19, $S' := S - f_1' + f_1$ satisfies the conditions in Proposition 7 or the auxiliary digraph H' associated with S' and T has a dicycle of length at most $q - 1$. Since the shortest dicycle length decreases monotonically, by applying such a transformation of S at most q times, we obtain a feasible arc subset S' satisfying the conditions in Proposition 7. This completes the proof of Proposition 7. Furthermore, by applying Proposition 7 p times, we obtain Theorem 1. Note that since all the proofs are constructive and each X_i can be computed by using a minimum s-t cut algorithm, the reconfiguration sequence can be computed in polynomial time.

4 Extension to Arborescences with Distinct Roots

In this section, we prove Theorem 2. That is, we extend Theorem 1 to the case when a feasible arc set is the union of k arc-disjoint arborescences that may have distinct roots.

Proof of Theorem 2. Extend V by adding a new vertex $\widehat{r}$. For an arc set $A' \subseteq A$ satisfying $\delta^-_{A'}(v) \leq k$ for each $v \in V$, let $\widehat{A'}$ denote the arc set of the digraph obtained from A' by adding $k - \delta^-_{A'}(v)$ parallel arcs from $\widehat{r}$ to v for each $v \in V$. Observe that $A' \in \mathcal{F}_k$ holds if and only if $\widehat{r}$ has outdegree k in $\widehat{A'}$ and $\widehat{A'}$ can be partitioned into k arc-disjoint $\widehat{r}$-arborescences on $V + \widehat{r}$. By Theorem 5, the latter is equivalent to that $\delta^-_{\widehat{A'}}(X) \geq k$ for any $X \subseteq V$ with $X \neq \emptyset$.

We consider the case first when the multisets of roots in the decompositions of S and T into k arc-disjoint arborescences are not the same.

▶ **Lemma 20.** *Suppose that $\delta^-_S(v) \neq \delta^-_T(v)$ holds for a vertex $v \in V$. Then there exist arcs $e \in S \setminus T$ and $f \in T \setminus S$ such that $S - e + f \in \mathcal{F}_k$.*

Proof. Since $\sum_{w \in V} \delta^-_S(w) = \sum_{w \in V} \delta^-_T(w)$, there is a vertex $v \in V$ with $\delta^-_S(v) < \delta^-_T(v)$. Then there exists an arc $f \in T \setminus S$ with $\mathrm{head}(f) = v$. Let X denote the unique minimal subset of V containing f which is tight with respect to $\widehat{S}$, i.e., $\delta^-_{\widehat{S}}(X) = k$. Note that such a tight set exists as $\delta^-_{\widehat{S}}(V) = k$. Since $\delta^-_{\widehat{S}}(w) = \delta^-_{\widehat{T}}(w) = k$ for any $w \in V$ and $\delta^-_{\widehat{T}}(X) \geq k = \delta^-_{\widehat{S}}(X)$,

$$\Delta_T(X, X) = k|X| - \delta^-_{\widehat{T}}(X) \leq k|X| - \delta^-_{\widehat{S}}(X) = \Delta_S(X, X).$$

Therefore, X contains an arc $e \in S \setminus T$, since it contains the arc $f \in T \setminus S$.

We claim that $S' = S - e + f \in \mathcal{F}_k$. Let $u = \mathrm{head}(e)$, then

$$\widehat{S'} = \widehat{S} - e + (\widehat{r}, u) + f - (\widehat{r}, v).$$

We easily see that $\delta^-_{S'}(w) \leq k$ holds for any $w \in V$, as $\delta^-_{S'}(v) \leq \delta^-_S(v) + 1 \leq \delta^-_T(v) \leq k$. Thus, it suffices to show that $\delta^-_{\widehat{S'}}(Z) \geq k$ holds for any nonempty subset $Z \subseteq V$. Assume to the contrary that there exists a nonempty subset $Z \subseteq V$ with $\delta^-_{\widehat{S'}}(Z) \leq k - 1$. Then

$$k \leq \delta^-_{\widehat{S}}(Z) \leq \delta^-_{\widehat{S} - e + (\widehat{r}, u)}(Z) \leq \delta^-_{\widehat{S'}}(Z) + 1 \leq (k-1) + 1 = k,$$

thus $\delta^-_{\widehat{S}}(Z) = k$ and $\delta^-_{\widehat{S}}(Z) = \delta^-_{\widehat{S} - e + (\widehat{r}, u)}(Z) = \delta^-_{\widehat{S'}}(Z) + 1$. These show that Z is a tight set with respect to $\widehat{S}$, it does not contain e, and it contains f. Since $X \cap Z$ is a tight set with respect to $\widehat{S}$ by Lemma 6 and $X \cap Z \subsetneq X$ as Z does not contain e, this contradicts the minimality of X. ◀

We turn to the proof of the theorem. By the repeated application of Lemma 20, there exists a sequence $T_0, T_1, \ldots, T_m$ such that $T_0 = S$, $\delta^-_{T_m}(v) = \delta^-_T(v)$ for any $v \in V$, $T_i \in \mathcal{F}_k$ for $i \in [m] \cup \{0\}$ and $|T_{i-1} \setminus T_i| = |T_i \setminus T_{i-1}| = 1$ for $i \in [m]$. By Theorem 1, there exists a sequence $T'_m, T'_{m+1}, \ldots, T'_\ell$ such that $T'_m = \widehat{T_m}$, $T'_\ell = \widehat{T}$, T'_i is a subset of $\widehat{T'_m} \cup \widehat{T}$ which can be partitioned into k arc-disjoint $\widehat{r}$-arborescences on $V + \widehat{r}$ for $i \in \{m, m+1, \ldots, \ell\}$, and $|T'_{i-1} \setminus T'_i| = |T'_i \setminus T'_{i-1}| = 1$ for $i \in \{m+1, m+2, \ldots, \ell\}$. Then for any $i \in \{m+1, m+2, \ldots, \ell\}$ there is an arc set $T_i \subseteq A$ such that $T'_i = \widehat{T_i}$, as $\delta^-_{T'_i}(v) = k$ for any $v \in V$. Since $T'_i \subseteq \widehat{T'_m} \cup \widehat{T}$ and T'_i can be partitioned into k arc-disjoint $\widehat{r}$-arborescences, $\widehat{r}$ has outdegree k in T'_i, and T_i can be partitioned into k arc-disjoint arborescences. Therefore, $T_i \in \mathcal{F}_k$ holds for $i \in \{m+1, m+2, \ldots, \ell\}$, hence the sequence $T_0, T_1, \ldots, T_\ell$ satisfies the properties required by the theorem. ◀

5 Proof of Theorem 4

In this section, we give a proof of Theorem 4, which we restate here.

▶ **Theorem 4.** *There exist matroids $M_1 = (E, \mathcal{B}_1)$ and $M_2 = (E, \mathcal{B}_2)$ such that the pair of matroids (M_1, M_2) satisfies (RCB), while $(2M_1, 2M_2)$ does not satisfy it.*

Proof. Consider the matroids $M_1 = (E, \mathcal{B}_1)$ and $M_2 = (E, \mathcal{B}_2)$ on ground set $E = \{a, b, c_1, c_2, c_3, d_1, d_2, d_3\}$ defined by their families of bases

$$\mathcal{B}_1 = \{B \subseteq E \mid |B| = 3, |B \cap \{c_1, c_2, c_3\}| \leq 1, |B \cap \{d_1, d_2, d_3\}| \leq 1\},$$
$$\mathcal{B}_2 = \{B \subseteq E \mid |B| = 3, |B \cap \{a, c_1, d_1\}| = 1\}.$$

Note that M_1 is the truncation of the direct sum of the uniform matroids of rank 1 on $\{a\}$, $\{b\}$, $\{c_1, c_2, c_3\}$, and $\{d_1, d_2, d_3\}$, while M_2 is the direct sum of the uniform matroid of rank 1 on $\{a, c_1, d_1\}$ and the uniform matroid of rank 2 on $\{b, c_2, c_3, d_2, d_3\}$.

We prove that the pair (M_1, M_2) satisfies (RCB). The common bases of M_1 and M_2 are the sets of the form

$$\{a, b, c_i\}, \{a, b, d_j\}, \{a, c_i, d_j\}, \{b, c_1, d_j\}, \{b, c_i, d_1\}$$

for $i, j \in \{2, 3\}$. It is enough to show the existence of a reconfiguration sequence between $\{b, c_1, d_2\}$ and each $B \in \mathcal{B}_1 \cap \mathcal{B}_2$. For $i, j \in \{2, 3\}$, consider the sequence of common bases

$$\{b, c_1, d_2\}, \{b, c_1, d_j\}, \{a, b, d_j\}, \{a, c_i, d_j\}, \{a, b, c_i\}, \{b, c_i, d_j\},$$

where we omit the second term for $j = 2$. This sequence starts from $\{b, c_1, d_2\}$, contains each $B \in \mathcal{B}_1 \cap \mathcal{B}_2$ for appropriate values of $i, j \in \{2, 3\}$, and $|B' \setminus B''| = |B'' \setminus B'| = 1$ holds for each pair of adjacent terms B', B'' of the sequence, thus it proves our claim.

Next we show that the matroids $2M_1 = (E, \mathcal{B}_1^2)$ and $2M_2 = (E, \mathcal{B}_2^2)$ do not satisfy (RCB). Recall that $2M_i$ is the matroid whose bases are the unions of two disjoint bases of M_i. We have

$$\mathcal{B}_1^2 = \{B \subseteq E \mid \{a, b\} \subseteq B, |B \cap \{c_1, c_2, c_3\}| = 2, |B \cap \{d_1, d_2, d_3\}| = 2\},$$
$$\mathcal{B}_2^2 = \{B \subseteq E \mid |B \cap \{a, c_1, d_1\}| = 2, |B \cap \{b, c_2, c_3, d_2, d_3\}| = 4\},$$

thus

$$\mathcal{B}_1^2 \cap \mathcal{B}_2^2 = \{\{a, b, c_1, c_i, d_2, d_3\} \mid i \in \{2, 3\}\} \cup \{\{a, b, c_2, c_3, d_1, d_j\} \mid j \in \{2, 3\}\}.$$

Since $|B \setminus B'| = |B' \setminus B| = 2$ for any $B \in \{\{a, b, c_1, c_i, d_2, d_3\} \mid i \in \{2, 3\}\}$ and $B' \in \{\{a, b, c_2, c_3, d_1, d_j\} \mid j \in \{2, 3\}\}$, the pair $(2M_1, 2M_2)$ does not satisfy (RCB). ◀

6 Concluding Remarks

In this paper, we showed the reconfigurability of the union of k arborescences for fixed k. In other words, we showed that the pair of matroids representing the union of k arborescences satisfies (RCB). It will be interesting to investigate whether (RCB) holds or not for other classes of matroid pairs, e.g., White's conjecture [28].

Another interesting topic is the length of a shortest reconfiguration sequence. For the union of k arborescences, in the full version of this paper, we give an upper bound on the length of a shortest reconfiguration sequence, which is slightly smaller than $k|S \setminus T|$.

Meanwhile, there is an example whose shortest length is $\frac{3}{2}|S \setminus T|$, which is obtained by combining many copies of the digraph in Example 3. It will be interesting if we can close the gap between these bounds. It is also open whether we can find a shortest reconfiguration sequence from S to T in polynomial time if S and T are given as input.

The length of a shortest reconfiguration sequence can be considered also for other classes of matroid pairs. When M_2 is the dual matroid of M_1, Hamidoune conjectured that there always exists a reconfiguration sequence whose length is at most the size of each common basis (or equivalently, the rank of the matroids); see [7]. This conjecture is stronger than White's conjecture [28], and is open even for some special cases, e.g. when M_1 is a graphic matroid and M_2 is its dual.

Polynomial solvability of Matroid Intersection Reconfiguration (see Section 1.3) is also an interesting and challenging open problem.

References

1 Francisco Barahona and William R Pulleyblank. Exact arborescences, matchings and cycles. *Discrete Applied Mathematics*, 16(2):91–99, 1987. doi:10.1016/0166-218x(87)90067-9.

2 Kristóf Bérczi and András Frank. Variations for Lovász' submodular ideas. In *Building Bridges*, pages 137–164, 2010. doi:10.1007/978-3-540-85221-6_4.

3 Kristóf Bérczi and Tamás Schwarcz. Exchange distance of basis pairs in split matroids. *arXiv preprint*, 2022. arXiv:2203.01779.

4 Joseph E. Bonin. Basis-exchange properties of sparse paving matroids. *Advances in Applied Mathematics*, 50(1):6–15, 2013. doi:10.1016/j.aam.2011.05.006.

5 Nicolas Bousquet, Takehiro Ito, Yusuke Kobayashi, Haruka Mizuta, Paul Ouvrard, Akira Suzuki, and Kunihiro Wasa. Reconfiguration of spanning trees with many or few leaves. In *28th Annual European Symposium on Algorithms (ESA 2020)*, pages 24:1–24:15, 2020. doi:10.4230/LIPIcs.ESA.2020.24.

6 Nicolas Bousquet, Takehiro Ito, Yusuke Kobayashi, Haruka Mizuta, Paul Ouvrard, Akira Suzuki, and Kunihiro Wasa. Reconfiguration of spanning trees with degree constraint or diameter constraint. In *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, pages 15:1–15:21, 2022. doi:10.4230/LIPIcs.STACS.2022.15.

7 Raul Cordovil and M. Leonor Moreira. Bases-cobases graphs and polytopes of matroids. *Combinatorica*, 13(2):157–165, 1993. doi:10.1007/bf01303201.

8 Olivier Durand de Gevigney, Viet-Hang Nguyen, and Zoltán Szigeti. Matroid-based packing of arborescences. *SIAM Journal on Discrete Mathematics*, 27(1):567–574, 2013. doi:10.1137/120883761.

9 Jack Edmonds. Edge-disjoint branchings. In *Combinatorial algorithms, Courant Computer Science Symposium 9*, pages 91–96. Algorithmics Press, New York, 1973.

10 Martin Farber, Bruce Richter, and Herbert Shank. Edge-disjoint spanning trees: A connectedness theorem. *Journal of Graph Theory*, 9(3):319–324, 1985. doi:10.1002/jgt.3190090303.

11 Quentin Fortier, Csaba Király, Marion Léonard, Zoltán Szigeti, and Alexandre Talon. Old and new results on packing arborescences in directed hypergraphs. *Discrete Applied Mathematics*, 242:26–33, 2018. doi:10.1016/j.dam.2017.11.004.

12 András Frank. On disjoint trees and arborescences. In *Algebraic Methods in Graph Theory*, pages 159–169. North-Holland, Amsterdam, 1978.

13 András Frank, Tamás Király, and Zoltán Király. On the orientation of graphs and hypergraphs. *Discrete Applied Mathematics*, 131(2):385–400, 2003. doi:10.1016/S0166-218X(02)00462-6.

14 Satoru Fujishige. A note on disjoint arborescences. *Combinatorica*, 30(2):247–252, 2010. doi:10.1007/s00493-010-2518-y.

15 Hui Gao and Daqing Yang. Packing of maximal independent mixed arborescences. *Discrete Applied Mathematics*, 289:313–319, 2021. doi:10.1016/j.dam.2020.11.009.

16 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(12-14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

17 Takehiro Ito, Yuni Iwamasa, Yasuaki Kobayashi, Yu Nakahata, Yota Otachi, and Kunihiro Wasa. Reconfiguring (non-spanning) arborescences. *Theoretical Computer Science*, 943:131–141, 2023. doi:10.1016/j.tcs.2022.12.007.

18 Takehiro Ito, Jun Kawahara, Yu Nakahata, Takehide Soh, Akira Suzuki, Junichi Teruyama, and Takahisa Toda. ZDD-based algorithmic framework for solving shortest reconfiguration problems. In Andre A. Cire, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 167–183. Springer, Cham, Switzerland, 2023. doi:10.1007/978-3-031-33271-5_12.

19 Naoyuki Kamiyama, Naoki Katoh, and Atsushi Takizawa. Arc-disjoint in-trees in directed graphs. *Combinatorica*, 29(2):197–214, 2009. doi:10.1007/s00493-009-2428-z.

20 Csaba Király. On maximal independent arborescence packing. *SIAM Journal on Discrete Mathematics*, 30(4):2107–2114, 2016. doi:10.1137/130938396.

21 Michał Lasoń and Mateusz Michałek. On the toric ideal of a matroid. *Advances in Mathematics*, 259:1–12, 2014. doi:10.1016/j.aim.2014.03.004.

22 László Lovász. On two minimax theorems in graph. *Journal of Combinatorial Theory, Series B*, 21(2):96–103, 1976. doi:10.1016/0095-8956(76)90049-6.

23 Tatsuya Matsuoka and Shin-ichi Tanigawa. On reachability mixed arborescence packing. *Discrete Optimization*, 32:1–10, 2019. doi:10.1016/j.disopt.2018.10.002.

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

25 James Oxley. *Matroid Theory*, volume 21 of *Oxford Graduate Texts in Mathematics*. Oxford University Press, Oxford, second edition, 2011.

26 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003.

27 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, Cambridge, UK, 2013. doi:10.1017/CBO9781139506748.005.

28 Neil L. White. A unique exchange property for bases. *Linear Algebra and its Applications*, 31:81–91, 1980. doi:10.1016/0024-3795(80)90209-8.

An Approximation Algorithm for Two-Edge-Connected Subgraph Problem via Triangle-Free Two-Edge-Cover

Yusuke Kobayashi
Research Institute for Mathematical Sciences, Kyoto University, Japan

Takashi Noguchi
Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

The 2-Edge-Connected Spanning Subgraph problem (2-ECSS) is one of the most fundamental and well-studied problems in the context of network design. We are given an undirected graph G, and the objective is to find a 2-edge-connected spanning subgraph H of G with the minimum number of edges. For this problem, a lot of approximation algorithms have been proposed in the literature. In particular, very recently, Garg, Grandoni, and Ameli gave an approximation algorithm for 2-ECSS with a factor of 1.326, which is the best approximation ratio. In this paper, under the assumption that a maximum triangle-free 2-matching can be found in polynomial time in a graph, we give a $(1.3+\varepsilon)$-approximation algorithm for 2-ECSS, where ε is an arbitrarily small positive fixed constant. Note that a complicated polynomial-time algorithm for finding a maximum triangle-free 2-matching is announced by Hartvigsen in his PhD thesis, but it has not been peer-reviewed or checked in any other way. In our algorithm, we compute a minimum triangle-free 2-edge-cover in G with the aid of the algorithm for finding a maximum triangle-free 2-matching. Then, with the obtained triangle-free 2-edge-cover, we apply the arguments by Garg, Grandoni, and Ameli.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Mathematics of computing → Combinatorial optimization

Keywords and phrases approximation algorithm, survivable network design, minimum 2-edge-connected spanning subgraph, triangle-free 2-matching

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.49

Related Version *Full Version*: `https://arxiv.org/abs/2304.13228`

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 JP20K11692 and JP22H05001.

1 Introduction

In the field of survivable network design, a basic problem is to construct a network with minimum cost that satisfies a certain connectivity constraint. A seminal result by Jain [13] provides a 2-approximation algorithm for a wide class of survivable network design problems. For specific problems among them, a lot of better approximation algorithms have been investigated in the literature.

In this paper, we study the 2-Edge-Connected Spanning Subgraph problem (2-ECSS), which is one of the most fundamental and well-studied problems in this context. In 2-ECSS, we are given an undirected graph $G = (V, E)$, and the objective is to find a 2-edge-connected spanning subgraph H of G with the minimum number of edges. It was shown in [4, 5] that 2-ECSS does not admit a PTAS unless P = NP. Khuller and Vishkin [14] gave a 3/2-approximation algorithm for this problem, which was the starting point of the study of approximation algorithms for 2-ECSS. Cheriyan, Sebő, and Szigeti [1] improved this ratio to

17/12. Later, Hunkenschröder, Vempala, and Vetta [12] gave a 4/3-approximation algorithm, which rectifies flaws in [22]. By a completely different approach, Sebő and Vygen [19] achieved approximation ratio and integrality gap of 4/3. Very recently, Garg, Grandoni, and Ameli [8] improved this ratio to 1.326 by introducing powerful reduction steps and developing the techniques in [12].

The contribution of this paper is to present a $(1.3 + \varepsilon)$-approximation algorithm for 2-ECSS for any $\varepsilon > 0$ under the assumption that a maximum 2-matching containing no cycle of length at most 3 (called a maximum *triangle-free* 2-matching) can be found in polynomial time in a graph.

▶ **Theorem 1.** *Assume that there exists a polynomial-time algorithm for finding a maximum 2-matching that contains no cycle of length at most 3 in a graph. Then, for any constant $\varepsilon > 0$, there is a polynomial-time $(1.3 + \varepsilon)$-approximation algorithm for 2-ECSS.*

Note that a complicated polynomial-time algorithm for finding a maximum triangle-free 2-matching is announced by Hartvigsen [10], which indicates that the assumption in Theorem 1 holds. However, since this result has not been peer-reviewed or checked in any other way, we have retained the assumption in Theorem 1.

Our algorithm and its analysis are heavily dependent on the well-developed arguments by Garg, Grandoni, and Ameli [8]. In our algorithm, we first apply the reduction steps given in [8]. Then, instead of a minimum 2-edge-cover, we compute a minimum triangle-free 2-edge-cover in the graph, which is the key ingredient in our algorithm. We show that this can be done in polynomial time with the aid of a polynomial algorithm for finding a maximum triangle-free 2-matching. Finally, we convert the obtained triangle-free 2-edge-cover into a spanning 2-edge-connected subgraph by using the arguments in [8].

Our main technical contribution is to point out the utility of a maximum triangle-free 2-matching in the arguments by Garg, Grandoni, and Ameli [8].

Related Work

A natural extension of 2-ECSS is the k-Edge-Connected Spanning Subgraph problem (k-ECSS), which is to find a k-edge-connected spanning subgraph of the input graph with the minimum number of edges. For k-ECSS, several approximation algorithms have been proposed, in which approximation factors depend on k [2, 6, 7]. We can also consider the weighted variant of 2-ECSS, in which the objective is to find a 2-edge-connected spanning subgraph with the minimum total weight in a given edge-weighted graph. The result of Jain [13] leads to a 2-approximation algorithm for the weighted 2-ECSS, and it is still the best known approximation ratio. For the case when all the edge weights are 0 or 1, which is called the *forest augmentation problem*, Grandoni, Ameli, and Traub [9] recently gave a 1.9973-approximation algorithm. Furthermore, for the *tree augmentation problem*, which is the case when 0-weight edges are connected, the approximation ratio was improved to $1.5 + \varepsilon$ for any $\varepsilon > 0$ in a series of works by Traub and Zenklusen [20, 21]. See references in [8, 9] for more related work on survivable network design problems.

It is well-known that a 2-matching of maximum size can be found in polynomial-time by using a matching algorithm; see e.g., [18, Section 30]. As a variant of this problem, the problem of finding a maximum 2-matching that contains no cycle of length at most k, which is called the *$C_{\leq k}$-free 2-matching problem*, has been actively studied. Hartvigsen [10] announced a polynomial-time algorithm for the $C_{\leq 3}$-free 2-matching problem (also called the *triangle-free 2-matching problem*), and Papadimitriou showed the NP-hardness for $k \geq 5$ (see [3]). The polynomial solvability of the $C_{\leq 4}$-free 2-matching problem has been open

for more than 40 years. The edge weighted variant of the $C_{\leq 3}$-free 2-matching problem is also a relevant open problem in this area, and some positive results are known for special cases [11, 15, 16, 17]. See references in [16] for more related work on the $C_{\leq k}$-free 2-matching problem.

2 Preliminaries

Throughout the paper, we only consider simple undirected graphs, i.e., every graph has neither self-loops nor parallel edges.[1] A graph $G = (V, E)$ is said to be 2-*edge-connected* if $G \setminus \{e\}$ is connected for every $e \in E$, and it is called 2-*vertex-connected* if $G \setminus \{v\}$ is connected for every $v \in V$ and $|V| \geq 3$. For a subgraph H of G, its vertex set and edge set are denoted by $V(H)$ and $E(H)$, respectively. A subgraph H of $G = (V, E)$ is *spanning* if $V(H) = V(G)$. In the 2-Edge-Connected Spanning Subgraph problem (2-ECSS), we are given a graph $G = (V, E)$ and the objective is to find a 2-edge-connected spanning subgraph H of G with the minimum number of edges (if one exists).

In this paper, a spanning subgraph H is often identified with its edge set $E(H)$. Let H be a spanning subgraph (or an edge set) of G. A connected component of H which is 2-edge-connected is called a *2EC component of* H. A 2EC component of H is called an *i-cycle 2EC component* if it is a cycle of length i. In particular, a 3-cycle 2EC component is called a *triangle 2EC component*. A maximal 2-edge-connected subgraph B of H is called a *block* of H if $|V(B)| \geq 3$ and B is not a 2EC component. An edge $e \in E(H)$ is called a *bridge* of H if $H \setminus \{e\}$ has more connected components than H. A block B of H is called a *leaf block* if H has exactly one bridge incident to B, and an *inner block* otherwise.

Let $G = (V, E)$ be a graph. For an edge set $F \subseteq E$ and a vertex $v \in V$, let $d_F(v)$ denote the number of edges in F that are incident to v. An edge set $F \subseteq E$ is called a 2-*matching* if $d_F(v) \leq 2$ for every $v \in V$, and it is called a 2-*edge-cover* if $d_F(v) \geq 2$ for every $v \in V$.[2]

3 Algorithm in Previous Work

Since our algorithm is based on the well-developed 1.326-approximation algorithm given by Garg, Grandoni, and Ameli [8], we describe some of their results in this section.

3.1 Reduction to Structured Graphs

In the algorithm by Garg, Grandoni, and Ameli [8], they first reduce the problem to the case when the input graph satisfies some additional conditions, where such a graph is called a $(5/4, \varepsilon)$-structured graph. In what follows in this paper, let $\varepsilon > 0$ be a sufficiently small positive fixed constant, which will appear in the approximation factor. In particular, we suppose that $0 < \varepsilon \leq 1/24$, which is used in the argument in [8]. We say that a graph $G = (V, E)$ is $(5/4, \varepsilon)$-*structured* if it is 2-vertex-connected, it contains at least $2/\varepsilon$ vertices, and it does not contain the following structures:

- **($5/4$-contractible subgraph)** a 2-edge-connected subgraph C of G such that every 2-edge-connected spanning subgraph of G contains at least $\frac{4}{5}|E(C)|$ edges with both endpoints in $V(C)$;
- **(irrelevant edge)** an edge $uv \in E$ such that $G \setminus \{u, v\}$ is not connected;

[1] It is shown in [12] that this assumption is not essential when we consider 2-ECSS.
[2] Such edge sets are sometimes called *simple* 2-matchings and *simple* 2-edge-covers in the literature.

- **(non-isolating 2-vertex-cut)** a vertex set $\{u, v\} \subseteq V$ of G such that $G \setminus \{u, v\}$ has at least three connected components or has exactly two connected components, both of which contain at least two vertices.

The following lemma shows that it suffices to consider $(5/4, \varepsilon)$-structured graphs when we design approximation algorithms.

▶ **Lemma 2** (Garg, Grandoni, and Ameli [8, Lemma 2.2]). *Let ε be a sufficiently small positive constant. For $\alpha \geq \frac{5}{4}$, if there exists a polynomial-time α-approximation algorithm for 2-ECSS on $(5/4, \varepsilon)$-structured graphs, then there exists a polynomial-time $(\alpha + 2\varepsilon)$-approximation algorithm for 2-ECSS.*

3.2 Semi-Canonical Two-Edge-Cover

A 2-edge-cover H of G (which is identified with a spanning subgraph) is called *semi-canonical* if it satisfies the following conditions.

(1) Each 2EC component of H is a cycle or contains at least 7 edges.

(2) Each leaf block contains at least 6 edges and each inner block contains at least 4 edges.

(3) There is no pair of edge sets $F \subseteq H$ and $F' \subseteq E \setminus H$ such that $|F| = |F'| \leq 3$, $(H \setminus F) \cup F'$ is a 2-edge-cover with fewer connected components than H, and F contains an edge in some triangle 2EC component of H.

(4) There is no pair of edge sets $F \subseteq H$ and $F' \subseteq E \setminus H$ such that $|F| = |F'| = 2$, $(H \setminus F) \cup F'$ is a 2-edge-cover with fewer connected components than H, both edges in F' connect two 4-cycle 2EC components, say C_1 and C_2, and F is contained in $C_1 \cup C_2$. In other words, by removing 2 edges and adding 2 edges, we cannot merge two 4-cycle 2EC components into a cycle of length 8.

▶ **Lemma 3** (Garg, Grandoni, and Ameli [8, Lemma 2.6]). *Let ε be a sufficiently small positive constant. Suppose we are given a semi-canonical 2-edge-cover H of a $(5/4, \varepsilon)$-structured graph G with $b|H|$ bridges and $t|H|$ edges belonging to triangle 2EC components of H. Then, in polynomial time, we can compute a 2-edge-connected spanning subgraph S of size at most $(\frac{13}{10} + \frac{1}{30}t - \frac{1}{20}b)|H|$.*

▶ Remark 4. In the original statement of [8, Lemma 2.6], H is assumed to satisfy a stronger condition than semi-canonical, called canonical. A 2-edge-cover H is said to be *canonical* if it satisfies (1) and (2) in the definition of semi-canonical 2-edge-covers, and also the following condition: there is no pair of edge sets $F \subseteq H$ and $F' \subseteq E \setminus H$ such that $|F| = |F'| \leq 3$ and $(H \setminus F) \cup F'$ is a 2-edge-cover with fewer connected components than H. However, one can see that the condition "canonical" can be relaxed to "semi-canonical" by following the proof of [8, Lemma 2.6]; see the proofs of Lemmas D.3, D.4, and D.11 in [8].

4 Algorithm via Triangle-Free Two-Edge-Cover

The idea of our algorithm is quite simple: we construct a semi-canonical 2-edge-cover H with no triangle 2EC components and then apply Lemma 3. We say that an edge set $F \subseteq E$ is *triangle-free* if there are no triangle 2EC components of F. Note that a triangle-free edge set F may contain a cycle of length three that is contained in a larger connected component. In order to construct a semi-canonical triangle-free 2-edge-cover, we use a polynomial-time algorithm for finding a triangle-free 2-matching given by Hartvigsen [10].

▶ **Theorem 5** (Hartvigsen [10, Theorem 3.2 and Proposition 3.4]). *For a graph G, we can find a triangle-free 2-matching in G with maximum cardinality in polynomial time.*

Note again that, since this result has not been published as a journal paper, we have retained the assumption in Theorem 1.

In Section 4.1, we give an algorithm for finding a minimum triangle-free 2-edge-cover with the aid of Theorem 5. Then, we transform it into a semi-canonical triangle-free 2-edge-cover in Section 4.2. Using the obtained 2-edge-cover, we give a proof of Theorem 1 in Section 4.3.

4.1 Minimum Triangle-Free Two-Edge-Cover

As with the relationship between 2-matchings and 2-edge-covers (see e.g. [18, Section 30.14]), triangle-free 2-matchings and triangle-free 2-edge-covers are closely related to each other, which can be stated as the following two lemmas.

▶ **Lemma 6.** *Let $G = (V, E)$ be a connected graph such that the minimum degree is at least two and $|V| \geq 4$. Given a triangle-free 2-matching M in G, we can compute a triangle-free 2-edge-cover C of G with size at most $2|V| - |M|$ in polynomial time.*

Proof. Starting with $F = M$, we perform the following update repeatedly while F is not a 2-edge-cover:

Choose a vertex $v \in V$ with $d_F(v) < 2$ and an edge $vw \in E \setminus F$ incident to v.

(i) If $F \cup \{vw\}$ contains no triangle 2EC components, then add vw to F.

(ii) Otherwise, $F \cup \{vw\}$ contains a triangle 2EC component with vertex set $\{u, v, w\}$ for some $u \in V$. In this case, choose an edge e connecting $\{u, v, w\}$ and $V \setminus \{u, v, w\}$, and add both vw and e to F.

If F becomes a 2-edge-cover, then the procedure terminates by returning $C = F$. It is obvious that this procedure terminates in polynomial steps and returns a triangle-free 2-edge-cover.

We now analyze the size of the output C. For an edge set $F \subseteq E$, define $g(F) = \sum_{v \in V} \max\{2 - d_F(v), 0\}$. Then, in each iteration of the procedure, we observe the following: in case (i), one edge is added to F and $g(F)$ decreases by at least one; in case (ii), two edges are added to F and $g(F)$ decreases by at least two, because $d_F(v) = d_F(w) = 1$ before the update. With this observation, we see that $|C| - |M| \leq g(M) - g(C) = \sum_{v \in V}(2 - d_M(v))$, where we note that M is a 2-matching and C is a 2-edge-cover. Therefore, it holds that

$$|C| \leq |M| + \sum_{v \in V}(2 - d_M(v)) = |M| + (2|V| - 2|M|) = 2|V| - |M|,$$

which completes the proof. ◀

▶ **Lemma 7.** *Given a triangle-free 2-edge-cover C in a graph $G = (V, E)$, we can compute a triangle-free 2-matching M of G with size at least $2|V| - |C|$ in polynomial time.*

Proof. Starting with $F = C$, we perform the following update repeatedly while F is not a 2-matching:

Choose a vertex $v \in V$ with $d_F(v) > 2$ and an edge $vw \in F$ incident to v.

(i) If $F \setminus \{vw\}$ contains no triangle 2EC components, then remove vw from F.

(ii) If $F \setminus \{vw\}$ contains a triangle 2EC component whose vertex set is $\{v, v_1, v_2\}$ for some $v_1, v_2 \in V$, then remove vv_1 from F.

(iii) If neither of the above holds, then $F \setminus \{vw\}$ contains a triangle 2EC component whose vertex set is $\{w, w_1, w_2\}$ for some $w_1, w_2 \in V$. In this case, remove ww_1 from F.

If F becomes a 2-matching, then the procedure terminates by returning $M = F$. It is obvious that this procedure terminates in polynomial steps and returns a triangle-free 2-matching.

We now analyze the size of the output M. For an edge set $F \subseteq E$, define $g(F) = \sum_{v \in V} \max\{d_F(v) - 2, 0\}$. Then, in each iteration of the procedure, we observe that one edge is removed from F and $g(F)$ decreases by at least one, where we note that $d_F(w) = 3$ before the update in case (iii). With this observation, we see that $|C| - |M| \leq g(C) - g(M) = \sum_{v \in V}(d_C(v) - 2)$, where we note that C is a 2-edge-cover and M is a 2-matching. Therefore, it holds that

$$|M| \geq |C| - \sum_{v \in V}(d_C(v) - 2) = |C| - (2|C| - 2|V|) = 2|V| - |C|,$$

which completes the proof. ◀

By using these lemmas and Theorem 5, we can compute a triangle-free 2-edge-cover with minimum cardinality in polynomial time.

▶ **Proposition 8.** *Suppose that a triangle-free 2-matching M with maximum cardinality in a graph can be found in polynomial time. Then, for a graph $G = (V, E)$, we can compute a triangle-free 2-edge-cover C of G with minimum cardinality in polynomial time (if one exists). Furthermore, $|C| = 2|V| - |M|$.*

Proof. It suffices to consider the case when G is a connected graph such that the minimum degree is at least two and $|V| \geq 4$. Let M be a triangle-free 2-matching in G with maximum cardinality, which can be computed in polynomial time by the assumption. Then, by Lemma 6, we can construct a triangle-free 2-edge-cover C of G with size at most $2|V| - |M|$.

We now show that G has no triangle-free 2-edge-cover C' with $|C'| < 2|V| - |M|$. Assume to the contrary that there exists a triangle-free 2-edge-cover C' of size smaller than $2|V| - |M|$. Then, by Lemma 7, we can construct a triangle-free 2-matching M' of G with size at least $2|V| - |C'|$. Since $|M'| \geq 2|V| - |C'| > 2|V| - (2|V| - |M|) = |M|$, this contradicts that M is a triangle-free 2-matching with maximum cardinality. Therefore, G has no triangle-free 2-edge-cover of size smaller than $2|V| - |M|$, which implies that C is a triangle-free 2-edge-cover with minimum cardinality. ◀

4.2 Semi-Canonical Triangle-Free Two-Edge-Cover

We show the following lemma saying that a triangle-free 2-edge-cover can be transformed into a semi-canonical triangle-free 2-edge-cover without increasing the size. Although the proof is almost the same as that of [8, Lemma 2.4], we describe it for completeness.

▶ **Lemma 9.** *Let ε be a sufficiently small positive constant. Given a triangle-free 2-edge-cover H of a $(5/4, \varepsilon)$-structured graph $G = (V, E)$, in polynomial time, we can compute a triangle-free 2-edge-cover H' of no larger size which is semi-canonical.*

Proof. Recall that an edge set is identified with the corresponding spanning subgraph of G. Starting with $H' = H$, while H' is not semi-canonical we apply one of the following operations in this order of priority. We note that H' is always triangle-free during the procedure, and hence it always satisfies condition (3) in the definition of semi-canonical 2-edge-cover.

(a) If there exists an edge $e \in H'$ such that $H' \setminus \{e\}$ is a triangle-free 2-edge-cover, then remove e from H'.

(b) If H' does not satisfy condition (4), then we merge two 4-cycle 2EC components into a cycle of length 8 by removing 2 edges and adding 2 edges. Note that the obtained edge set is a triangle-free 2-edge-cover that has fewer connected components.

(c) Suppose that condition (1) does not hold, i.e., there exists a 2EC component C of H' with fewer than 7 edges that is not a cycle. Since C is 2-edge-connected and not a cycle, we obtain $|E(C)| \geq |V(C)| + 1$. If $|V(C)| = 4$, then C contains at least 5 edges and contains a cycle of length 4, which contradicts that (a) is not applied. Therefore, $|V(C)| = 5$ and $|E(C)| = 6$. Since operation (a) is not applied, C is either a bowtie (i.e., two triangles that share a common vertex) or a $K_{2,3}$; see figures in the proof of [8, Lemma 2.4].

(c1) Suppose that C is a bowtie that has two triangles $\{v_1, v_2, u\}$ and $\{v_3, v_4, u\}$. If G contains an edge between $\{v_1, v_2\}$ and $\{v_3, v_4\}$, then we can replace C with a cycle of length 5, which decreases the size of H'. Otherwise, by the 2-vertex-connectivity of G, there exists an edge $zw \in E \setminus H'$ such that $z \in V \setminus V(C)$ and $w \in \{v_1, v_2, v_3, v_4\}$. In this case, we replace H' with $(H' \setminus \{uw\}) \cup \{zw\}$. Then, the obtained edge set is a triangle-free 2-edge-cover with the same size, which has fewer connected components.

(c2) Suppose that C is a $K_{2,3}$ with two sides $\{v_1, v_2\}$ and $\{w_1, w_2, w_3\}$. If every w_i has degree exactly 2, then every feasible 2-edge-connected spanning subgraph contains all the edges of C, and hence C is a $\frac{5}{4}$-contractible subgraph, which contradicts the assumption that G is $(5/4, \varepsilon)$-structured. If G contains an edge $w_i w_j$ for distinct $i, j \in \{1, 2, 3\}$, then we can replace C with a cycle of length 5, which decreases the size of H'. Otherwise, since some w_i has degree at least 3, there exists an edge $w_i u \in E \setminus H'$ such that $i \in \{1, 2, 3\}$ and $u \in V \setminus V(C)$. In this case, we replace H' with $(H' \setminus \{v_1 w_i\}) \cup \{w_i u\}$. Then, the obtained edge set is a triangle-free 2-edge-cover with the same size, which has fewer connected components.

(d) Suppose that the first half of condition (2) does not hold, i.e., there exists a leaf block B that has at most 5 edges. Let v_1 be the only vertex in B such that all the edges connecting $V(B)$ and $V \setminus V(B)$ are incident to v_1. Since operation (a) is not applied, we see that B is a cycle of length at most 5. Let $v_1, \ldots, v_\ell$ be the vertices of B that appear along the cycle in this order. We consider the following cases separately; see figures in the proof of [8, Lemma 2.4].

(d1) Suppose that there exists an edge $zw \in E \setminus H'$ such that $z \in V \setminus V(B)$ and $w \in \{v_2, v_\ell\}$. In this case, we replace H' with $(H' \setminus \{v_1 w\}) \cup \{zw\}$.

(d2) Suppose that v_2 and v_ℓ are adjacent only to vertices in $V(B)$ in G, which implies that $\ell \in \{4, 5\}$. If $v_2 v_\ell \notin E$, then every feasible 2EC spanning subgraph contains four edges (incident to v_2 and v_ℓ) with both endpoints in $V(B)$, and hence B is a $\frac{5}{4}$-contractible subgraph, which contradicts the assumption that G is $(5/4, \varepsilon)$-structured. Thus, $v_2 v_\ell \in E$. Since there exists an edge connecting $V \setminus V(B)$ and $V(B) \setminus \{v_1\}$ by the 2-vertex-connectivity of G, without loss of generality, we may assume that G has an edge $v_3 z$ with $z \in V \setminus V(B)$. In this case, we replace H' with $(H' \setminus \{v_1 v_\ell, v_2 v_3\}) \cup \{v_3 z, v_2 v_\ell\}$.

In both cases, the obtained edge set is a triangle-free 2-edge-cover with the same size. Furthermore, we see that either (i) the obtained edge set has fewer connected components or (ii) it has the same number of connected components and fewer bridges.

(e) Suppose that the latter half of condition (2) does not hold, i.e., there exists an inner block B that has at most 3 edges. Then, B is a triangle. Let $\{v_1, v_2, v_3\}$ be the vertex set of B. If there are at least two bridge edges incident to distinct vertices in $V(B)$, say wv_1

and zv_2, then edge v_1v_2 has to be removed by operation (a), which is a contradiction. Therefore, all the bridge edges in H' incident to B are incident to the same vertex $v \in V(B)$. In this case, we apply the same operation as (d).

We can easily see that each operation above can be done in polynomial time. We also see that each operation decreases the lexicographical ordering of $(|H'|, \mathrm{cc}(H'), \mathrm{br}(H'))$, where $\mathrm{cc}(H')$ is the number of connected components in H' and $\mathrm{br}(H')$ is the number of bridges in H'. This shows that the procedure terminates in polynomial steps. After the procedure, H' is a semi-canonical triangle-free 2-edge-cover with $|H'| \leq |H|$, which completes the proof. ◀

4.3 Proof of Theorem 1

By Lemma 2, in order to prove Theorem 1, it suffices to give a $\frac{13}{10}$-approximation algorithm for 2-ECSS in $(5/4, \varepsilon)$-structured graphs for a sufficiently small fixed $\varepsilon > 0$. Let $G = (V, E)$ be a $(5/4, \varepsilon)$-structured graph. By Proposition 8, we can compute a minimum-size triangle-free 2-edge-cover H of G in polynomial-time. Note that the optimal value OPT of 2-ECSS in G is at least $|H|$, because every feasible solution for 2-ECSS is a triangle-free 2-edge-cover. By Lemma 9, H can be transformed into a semi-canonical triangle-free 2-edge-cover H' with $|H'| \leq |H|$. Since H' is triangle-free, by applying Lemma 3 with H', we obtain a 2-edge-connected spanning subgraph S of size at most $(\frac{13}{10} - \frac{1}{20}b)|H'|$, where H' has $b|H'|$ bridges. Therefore, we obtain

$$|S| \leq \left(\frac{13}{10} - \frac{1}{20}b\right)|H'| \leq \frac{13}{10}|H| \leq \frac{13}{10}\mathsf{OPT},$$

which shows that S is a $\frac{13}{10}$-approximate solution for 2-ECSS in G. This completes the proof of Theorem 1. ◀

5 Concluding Remarks

In this paper, we have presented a $(1.3 + \varepsilon)$-approximation algorithm (for any $\varepsilon > 0$) for 2-ECSS under the assumption that a maximum triangle-free 2-matching can be found in polynomial time. If the correctness of Theorem 5 is acknowledged, then our result achieves the best approximation ratio.

We conclude this paper by showing that the assumption in our main result (Theorem 1) can be relaxed with the aid of the following proposition.

▶ **Proposition 10.** *Let $0 < \alpha < 1$. Given a $(1-\alpha)$-approximate solution M' of a maximum triangle-free 2-matching problem in a graph $G = (V, E)$, we can compute a $(1+\alpha)$-approximate solution of the minimum triangle-free 2-edge-cover problem in G in polynomial time (if one exists).*

Proof. Let M and C be a maximum triangle-free 2-matching and a minimum 2-edge-cover in G, respectively. By Proposition 8, it holds that $|C| = 2|V| - |M|$. Given a $(1-\alpha)$-approximate solution M' of a maximum triangle-free 2-matching problem, by Lemma 6, we can construct a triangle-free 2-edge-cover C' in G with size at most $2|V| - |M'|$. Then, we have

$$\begin{aligned} |C'| &\leq 2|V| - |M'| \leq 2|V| - (1-\alpha)|M| \\ &= 2|V| - |M| + \alpha|M| \leq |C| + \alpha|C| = (1+\alpha)|C|, \end{aligned}$$

where we note that $|M| \leq |V| \leq |C|$ as M is a 2-matching and C is a 2-edge-cover. This shows that C' is a $(1+\alpha)$-approximate solution of the minimum triangle-free 2-edge-cover problem. ◀

By using this proposition instead of Proposition 8, we obtain the following theorem in the same way as Theorem 1.

▶ **Theorem 11.** *Assume that, for any $\varepsilon' > 0$, there exists a $(1-\varepsilon')$-approximation algorithm for finding a maximum 2-matching that contains no cycle of length at most 3 in a graph. Then, for any constant $\varepsilon > 0$, there is a polynomial-time $(1.3+\varepsilon)$-approximation algorithm for 2-ECSS.*

This theorem suggests that an approximate solution for the maximum triangle-free 2-matching problem is sufficient for our purpose. Therefore, it will be interesting to give a simple PTAS for the maximum triangle-free 2-matching problem.

References

1. Joseph Cheriyan, András Sebő, and Zoltán Szigeti. Improving on the 1.5-approximation of a smallest 2-edge connected spanning subgraph. *SIAM Journal on Discrete Mathematics*, 14(2):170–180, 2001. doi:10.1137/S0895480199362071.
2. Joseph Cheriyan and Ramakrishna Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM Journal on Computing*, 30(2):528–560, 2000. doi:10.1137/S009753979833920X.
3. Gérard Cornuéjols and William Pulleyblank. A matching problem with side conditions. *Discrete Mathematics*, 29(2):135–159, 1980. doi:10.1016/0012-365x(80)90002-3.
4. Artur Czumaj and Andrzej Lingas. On approximability of the minimum-cost k-connected spanning subgraph problem. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*, pages 281–290, 1999.
5. Cristina G Fernandes. A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem. *Journal of Algorithms*, 28(1):105–124, 1998. doi:10.1006/jagm.1998.0931.
6. Harold N. Gabow and Suzanne R. Gallagher. Iterated rounding algorithms for the smallest k-edge connected spanning subgraph. *SIAM Journal on Computing*, 41(1):61–103, 2012. doi:10.1137/080732572.
7. Harold N. Gabow, Michel X. Goemans, Éva Tardos, and David P. Williamson. Approximating the smallest k-edge connected spanning subgraph by LP-rounding. *Networks*, 53(4):345–357, 2009. doi:10.1002/net.20289.
8. Mohit Garg, Fabrizio Grandoni, and Afrouz Jabal Ameli. Improved approximation for two-edge-connectivity. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 2368–2410, 2023. doi:10.1137/1.9781611977554.ch92.
9. Fabrizio Grandoni, Afrouz Jabal Ameli, and Vera Traub. Breaching the 2-approximation barrier for the forest augmentation problem. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*, pages 1598–1611, 2022. doi:10.1145/3519935.3520035.
10. David Hartvigsen. *Extensions of Matching Theory*. PhD thesis, Carnegie Mellon University, 1984. Available at https://david-hartvigsen.net.
11. David Hartvigsen and Yanjun Li. Polyhedron of triangle-free simple 2-matchings in subcubic graphs. *Mathematical Programming*, 138:43–82, 2013.
12. Christoph Hunkenschröder, Santosh Vempala, and Adrian Vetta. A 4/3-approximation algorithm for the minimum 2-edge connected subgraph problem. *ACM Transactions on Algorithms*, 15(4):1–28, 2019. doi:10.1145/3341599.
13. Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21:39–60, 1998. doi:10.1007/s004930170004.
14. Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41(2):214–235, 1994. doi:10.1145/174652.174654.

15 Yusuke Kobayashi. A simple algorithm for finding a maximum triangle-free 2-matching in subcubic graphs. *Discrete Optimization*, 7:197–202, 2010. doi:10.1016/j.disopt.2010.04.001.

16 Yusuke Kobayashi. Weighted triangle-free 2-matching problem with edge-disjoint forbidden triangles. *Mathematical Programming*, 192(1):675–702, 2022. doi:10.1007/s10107-021-01661-y.

17 Katarzyna Paluch and Mateusz Wasylkiewicz. A simple combinatorial algorithm for restricted 2-matchings in subcubic graphs - via half-edges. *Information Processing Letters*, 171:106146, 2021. doi:10.1016/j.ipl.2021.106146.

18 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003.

19 András Sebő and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-tsp, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014. doi:10.1007/s00493-014-2960-3.

20 Vera Traub and Rico Zenklusen. A better-than-2 approximation for weighted tree augmentation. In *Proceedings of the IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS 2021)*, pages 1–12, 2022. doi:10.1109/FOCS52979.2021.00010.

21 Vera Traub and Rico Zenklusen. A $(1.5+\varepsilon)$-approximation algorithm for weighted connectivity augmentation. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*, STOC 2023, pages 1820–1833, 2023. doi:10.1145/3564246.3585122.

22 Santosh Vempala and Adrian Vetta. Factor 4/3 approximations for minimum 2-connected subgraphs. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2000)*, pages 262–273, 2000. doi:10.1007/3-540-44436-X_26.

On Min-Max Graph Balancing with Strict Negative Correlation Constraints

Ting-Yu Kuo
Dept. of Computer Science, National Yang-Ming Chiao-Tung University, Hsinchu, Taiwan

Yu-Han Chen
Dept. of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan

Andrea Frosini
Dept. of Mathematics and Informatics, University of Florence, Italy

Sun-Yuan Hsieh
Dept. of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan
Dept. of Computer Science and Information Engineering, National Chi-Nan University, Puli, Taiwan

Shi-Chun Tsai
Dept. of Computer Science, National Yang-Ming Chiao-Tung University, Hsinchu, Taiwan

Mong-Jen Kao
Dept. of Computer Science, National Yang-Ming Chiao-Tung University, Hsinchu, Taiwan

Abstract

We consider the min-max graph balancing problem with strict negative correlation (SNC) constraints. The graph balancing problem arises as an equivalent formulation of the classic unrelated machine scheduling problem, where we are given a hypergraph $G = (V, E)$ with vertex-dependent edge weight function $p\colon E \times V \mapsto \mathbb{Z}^{\geq 0}$ that represents the processing time of the edges (jobs). The SNC constraints, which are given as edge subsets $C_1, C_2, \ldots, C_k$, require that the edges in the same subset cannot be assigned to the same vertex at the same time. Under these constraints, the goal is to compute an edge orientation (assignment) that minimizes the maximum workload of the vertices.

In this paper, we conduct a general study on the approximability of this problem. First, we show that, in the presence of SNC constraints, the case with $\max_{e\in E} |e| = \max_i |C_i| = 2$ is the only case for which approximation solutions can be obtained. Further generalization on either direction, e.g., $\max_{e\in E} |e|$ or $\max_i |C_i|$, will directly make computing a feasible solution an NP-complete problem to solve. Then, we present a 2-approximation algorithm for the case with $\max_{e\in E} |e| = \max_i |C_i| = 2$, based on a set of structural simplifications and a tailored assignment LP for this problem. We note that our approach is general and can be applied to similar settings, e.g., scheduling with SNC constraints to minimize the weighted completion time, to obtain similar approximation guarantees.

Further cases are discussed to describe the landscape of the approximability of this prbolem. For the case with $|V| \leq 2$, which is already known to be NP-hard, we present a fully-polynomial time approximation scheme (FPTAS). On the other hand, we show that the problem is at least as hard as vertex cover to approximate when $|V| \geq 3$.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Unrelated Scheduling, Graph Balancing, Strict Correlation Constraints

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.50

Funding *Shi-Chun Tsai*: Supported in part by National Science and Technology Council (NSTC), Taiwan, under Grants 112-2634-F-A49-001-MBK.
Mong-Jen Kao: Supported in part by National Science and Technology Council (NSTC), Taiwan, under Grants 111-2221-E-A49-118-MY3, 112-2628-E-A49-017-MY3, and 112-2634-F-A49-001-MBK.

1 Introduction

In the min-max graph balancing problem with strict negative correlation (SNC) constraints, we are given an edge-weighted hypergraph $G = (V, E)$ with edge weight function $p : E \times V \mapsto \mathbb{Z}^{\geq 0}$ and a collection of edge subsets $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$. An edge orientation (assignment) is a function σ that maps each edge to one of its endpoints, i.e., $\sigma(e) \in e$ for all $e \in E$, and the orientation is said to be feasible if, for any $1 \leq i \leq k$, there exists $e, e' \in C_i$ such that $\sigma(e) \neq \sigma(e')$, i.e., not all edges in C_i are assigned to the same vertex. The workload of a vertex $v \in V$ is defined to be the total weight of the edges assigned to it, i.e., $\sum_{e \in E \text{ s.t. } \sigma(e)=v} p_{e,v}$. The goal of this problem is to compute a feasible edge orientation that minimizes the maximum workload of the vertices.

The graph balancing problem is an equivalent formulation of the classic unrelated machine scheduling problem [22], where the edges in E are interpreted as jobs, the vertices V are the machines, and the weights of edges are the processing times of the jobs. In the following, we start with an introduction on the unrelated scheduling problem.

Lenstra et al. [22] presented an elegant LP-rounding scheme that exploits the extreme point structure and obtained a 2-approximation for the unrelated scheduling problem. They also showed that $(1.5 - \epsilon)$-approximation for any $\epsilon > 0$ is NP-hard to obtain. Since then, there has been no significant progress on the upper-bound nor lower-bound for this problem, and closing the gap is known as a major open problems in this field for over 30 years [26, 29].

It is worth noting that, even the strongest LP formulation ever known for this problem, i.e., the configuration LP [27, 4], has an integrality gap of 2 for this problem [28]. Due to the above reasons, subsequent research has mostly focused on restricted cases of the problem.

An important subcase that is widely considered in the literature is the restricted assignment case, which considers the vertex-independent edge weight function $p\colon E \mapsto \mathbb{Z}^{\geq 0}$. Svensson [27] showed that the configuration-LP has an integrality gap at most $33/17 \approx 1.9412$ for this problem. This bound was later improved to $11/6 \approx 1.833$ by Jansen and Rohwedder [21]. In terms of approximation guarantees, Chakrabarty et al. [9] showed that, when there are only two different types of edge weights, a $(2 - \delta)$-approximation can be obtained for some small fixed constant $\delta > 0$.

For restricted assignment case without hyperedges, i.e. $p\colon E \mapsto \mathbb{Z}^{\geq 0}$ and $\max_{e \in E} |e| = 2$, Ebenlendr et al. [16] presented a 1.75-approximation algorithm. They also showed that, even for this case, a $(1.5 - \epsilon)$-approximation is still NP-hard to obtain. Moreover, the hardness result in [16] holds when there are only two different types of edge weights. For this seemingly simple case, a 1.5-approximation can be obtained [19, 23, 10]. Interestingly, this is the only nontrivial special case of unrelated scheduling for which the exact approximability is known.

Our motivation for studying the SNC constraints originates from the growing attention on the pairwise negative correlation between jobs to surpass the long-standing guarantees for job scheduling to minimize the weighted completion time [3, 5]. In our setting, we consider the extreme case for which the negative correlation between jobs in the same group is one.

In general, the presence of SNC constraints makes the problem much harder to consider. Consider the *constraint graph* $G_{\mathcal{C}} := (E, \mathcal{C})$ with the edges in E being the vertices and the constraints in $\mathcal{C}$ being the hyperedges. Even for the case that G is a complete hypergraph, i.e., $e = V$ for all $e \in E$, determining whether or not G has a feasible edge orientation is already equivalent to the problem of determining whether or not $G_{\mathcal{C}}$ has a $|V|$-coloring such that no constraint in $\mathcal{C}$ is monochromatic. As graph coloring is NP-hard, determining the existence of feasible edge orientation in the presence of SNC constraints is in general NP-hard.

There are essentially two directions to bypass the inherent hardness of the SNC constraints. The first one is to assume that a feasible coloring for $G_{\mathcal{C}}$ is given in advance, e.g., [7], and the other is to consider restricted classes of $G_{\mathcal{C}}$ for which a feasible orientation is polynomial-time computable, e.g., [20, 13, 24, 25]. Notably, most of these works assumed restricted assignment case with complete hypergraph, i.e., $p\colon E \mapsto \mathbb{Z}^{\geq 0}$ and $e = V$ for all $e \in E$, which is known as the identical machine scheduling case in the literature, with special constraint graphs with $|C_i| = 2$ for all $1 \leq i \leq k$.

In the following we introduce the above results in more detail. Bodlaender et al. [7] showed that, when a χ-coloring for the constraint graph $G_{\mathcal{C}}$ is given in advance, a $(\chi+2)/2$-approximation can be obtained when $\chi \leq |V| - 1$, and a 3-approximation can be obtained when $\chi \leq |V|/2 + 1$. This is achieved by partitioning the vertices into χ groups in a way such that no SNC constraints exist for each group. Different approximation guarantees are obtained, based on different heuristics to distribute the number of vertices for each color.

Jansen et al. [20] considered the case for which $G_{\mathcal{C}}$ is a complete multipartite graph, i.e., the edges in E are partitioned into multiple groups, and each vertex must handle edges that are within the same group. For this case, they provided a polynomial-time approximation scheme (PTAS). Pikies et al. [25] further considered the unrelated scheduling case and gave a $(1+\epsilon)\overline{p}$-approximation for any $\epsilon > 0$, where $\overline{p} = \max_{e,v} p_{e,v} / \min_{e,v} p_{e,v}$ is the maximum ratio between the edge weights. This is done by ignoring the edge weights and applying the algorithm of Jansen et al. [20]. Surprisingly, this straightforward algorithm is proven to be tight. They also showed that, even when $G_{\mathcal{C}}$ is complete bipartite, an $O(n^b \overline{p}^{1-c})$-approximation is NP-hard to obtain for any $b, c > 0$.

Das and Wiese [13] considered the case for which $G_{\mathcal{C}}$ is a collection of cliques, i.e., none of the edges from the same clique can be assigned to the same machine. For this case, they achieved a PTAS for identical machine scheduling. For unrelated machine scheduling, they proved a $(\log n)^{1/4}$-inapproximability unless $\mathsf{NP} \subseteq \mathsf{ZPTIME}(2^{(\log n)^{O(1)}})$. For the positive side, Page and Solis-Oba [24] provided a b-approximation, where b is the number of cliques in $G_{\mathcal{C}}$. They also gave a $b/2$-approximation for the restricted case that $\max_{e \in E} |e| = 2$.

Further related works

A problem directly related to min-max graph balancing is the max-min fair allocation, for which the goal is to maximize the minimum workload of the machines under the same set of inputs [6, 4]. For the unrelated scheduling case, i.e., $p\colon E \times V \mapsto \mathbb{Z}^{\geq 0}$, it is known that, $(2-\epsilon)$-approximation for any $\epsilon > 0$ is NP-hard to obtain [6]. For any $\epsilon = \Omega(\log\log n / \log n)$, Chakrabarty et al. [8] provided an $O(n^{\epsilon})$-approximation in $O(n^{1/\epsilon})$-time. Furthermore, it is known that, the integrality gap of configuration LP is $\Omega(\sqrt{|V|})$ even when $|E| = O(|V|)$ [4].

For the restricted assignment case, i.e., $p\colon E \mapsto \mathbb{Z}^{\geq 0}$, it is known that $(2-\epsilon)$-approximation is still NP-hard to obtain [6]. Bansal and Sviridenko [4] presented an $O(\log\log m / \log\log\log m)$-approximation based on rounding the configuration LP. In a series of follow-up works [17, 2, 12] and a very recent work due to Haxell and Szabó [18], the integrality gap of configuration LP for this case is narrowed down to 3.534.

The first constant factor approximation for this case was obtained by Annamalai et al. [1], which introduced the concept of lazy updates on the algorithm of [2] for polynomial-time termination. The approximation guarantee was further improved to 6 [11, 14] and then 4 [12], by further deriving more structures for the local search algorithm. For the case for which $\max_e |e| = 2$, a tight 2-approximation can be obtained [8]. Notably, this is also the only special case for which the exact approximability is known for max-min fair allocation.

Our Results and Contributions

In this paper, we study the complexity of unrelated graph balancing problem with SNC constraints and provide a clear landscape on the approximability of this problem with respect to different structures of input graphs. In contrast to the previous works, e.g., [20, 25, 13, 24], which mostly considered SNC constraints with special structures, we always keep SNC constraints in its most general form and discuss the complexity of the problem.

First, we show that, in the most general setting, either $\max_{e\in E}|e| \geq 3$ or $\max_i |C_i| \geq 3$ directly makes it NP-hard to even determine the existence of a feasible solution for the input instance. Hence, the case that $\max_{e\in E}|e| = \max_i |C_i| = 2$ is the only case for which approximation solutions can be obtained in terms of polynomial-time computations.

Even for the case $\max_{e\in E}|e| = \max_i |C_i| = 2$, determining the feasibility of the input instance is still not a trivial task to accomplish. For this, we provide a characterization of infeasible instances that can be checked in polynomial-time. This is done by transforming the problem into an *implication graph* between the assignments.

Then, we present a 2-approximation algorithm for the case with $\max_{e\in E}|e| = \max_i |C_i| = 2$. Our ingredient for this part is LP-rounding that further exploits the implication between assignments. We transform the concept into a directed acyclic graph (DAG), for which we design a specific assignment LP. We provide a threshold-based rounding, which follows the topological ordering of the DAG. The feasibility of the rounded solution is then ensured by the DAG structure.

Note that, even when there is no SNC constraint, the ratio of 2 is still the best approximation guarantee known for the case with $\max_{e\in E}|e| = 2$. We also remark that, our approach is general and can be directly applied to similar problems, e.g., scheduling with SNC constraints to minimize the weighted completion time, to obtain a 2-approximation guarantee.

It is also worth noting that, the techniques by Ebenlendr et al. [16] and Chakrabarty et al. [8], which are used to obtain approximation results for the restricted assignment case with no SNC constraints, do not seem to be applicable here. A key step in their rounding algorithms is to fractionally-round a cycle for G while keeping the remaining assignments unchanged. With the SNC constraints in place, such a rounding step is not guaranteed.

To compose a complete landscape for this problem, further special cases for G are discussed. For the case when $|V| \leq 2$, we show that a fully polynomial-time approximation scheme (FPTAS) can be obtained, based on a pseudo-polynomial time dynamic programming algorithm. Note that this case already contains the partition problem as its special case and is NP-hard to solve. On the other hand, we show that the problem is at least as hard as vertex cover to approximate when $|V| \geq 3$. Hence, assuming the unique game conjecture, our approximation result is already tight for this case.

Organization of this Paper

The rest of this paper is organized as follows. In Section 2, we provide the hardness result when $\max_{e\in E}|e| \geq 3$ or $\max_i |C_i| \geq 3$. In Section 3, we present a characterization for infeasible instances and our 2-approximation algorithm. We provide our FPTAS for $|V| \leq 2$ and the hardness results for $|V| \geq 3$ in Section 4.

2 Preliminaries

In the min-max SNC-graph balancing problem, we are given a tuple $\Psi = (G = (V, E),\ p,\ \mathcal{C})$, where $G = (V, E)$ is a hypergraph, $p : E \times V \mapsto \mathbb{Z}^{\geq 0}$ is a vertex-dependent edge weight function, and $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ is a collection of edge subsets that is referred to as the SNC constraints.

An edge orientation (assignment) is a function σ that maps each edge to one of its endpoints, i.e., $\sigma(e) \in e$ for all $e \in E$, and the orientation σ is said to be feasible if, for any $1 \leq i \leq k$, there exists $e, e' \in C_i$ such that $\sigma(e) \neq \sigma(e')$, i.e., not all edges in C_i are assigned to the same vertex. The workload of a vertex $v \in V$ w.r.t. σ is defined to be the total weight of the edges assigned to it, i.e., $\sum_{e \in E \text{ s.t. } \sigma(e)=v} p_{e,v}$. The goal of this problem is to compute a feasible edge orientation that minimizes the maximum workload of the vertices.

Let $\Psi = (G = (V, E),\ p,\ \mathcal{C})$ be an instance of the min-max SNC-graph balancing. For any edge orientation σ, we will use $\sigma^{-1}(v) := \{e \in E \mid \sigma(e) = v\}$ for any $v \in V$ to denote the set of edges that are assigned to v.

We say that Ψ is an (α, β)-instance if $\max_{e \in E} |e| = \alpha$ and $\max_{C \in \mathcal{C}} |C| = \beta$. In the min-max (α, β)-SNC graph balancing problem, we assume that Ψ is an (α, β)-instance.

Complexity of Min-Max (α, β)-SNC Graph Balancing

In the following, we show that, when $\max(\alpha, \beta) \geq 3$, determining whether or not an (α, β) instance has a feasible solution is already an NP-hard problem. Hence, min-max $(2, 2)$-SNC graph balancing is the only case for which an approximation solution can be obtained in terms of polynomial-time computations.

For this, we consider the cases $\alpha \geq 3$ and $\beta \geq 3$ separately and construct NP-hard reductions for them. We note that, as the weight function p plays no role in determining the feasibility of the instance, we will omit the construction detail for p.

First, for the case $\alpha \geq 3$, we make a reduction from the 3-SAT problem. Let $\varphi = \{c_1, c_2, \ldots, c_m\}$ be a set of m clauses over n variables $x_1, \ldots, x_n$. We construct an instance $\Psi = (G = (V, E), p, \mathcal{C})$ with $\max_{e \in E} |e| \leq 3$ as follows. For each variable x_i, we create two literal vertices v_{x_i} and $v_{\neg x_i}$ and an edge $e_{x_i} = \{v_{x_i}, v_{\neg x_i}\}$. Intuitively, this edge is supposed to be oriented to the negated value of x_i in a satisfying assignment, i.e., e_{x_i} should be oriented to $v_{\neg x_i}$ if x_i is true in a satisfying assignment and vice versa.

For each clause c_j, we construct a hyperedge e_{c_j} which contains the three literal vertices that c_j contains. Furthermore, for each clause c_j and each variable, say, x_i, that appears in c_j, we create an SNC constraint $C_{j,i} = \{e_{c_j}, e_{x_i}\}$. Intuitively, the hyperedge e_{c_j} for each clause c_j is supposed to be oriented to one of the literals that is true in a satisfying assignment, and the consistency between the orientations of the variables and clauses is provided by the SNC constraints we created. We have the following lemma.

▶ **Lemma 1.** *φ is satisfiable if and only if there exists a feasible orientation for Ψ.*

For the case $\beta \geq 3$, we make a reduction from 3-uniform hypergraph 2-coloring [15]. We show that, the problem of computing a feasible orientation for $(2, 3)$-SNC graph balancing already contains the 3-uniform hypergraph 2-coloring problem as one of its special cases.

Recall that, in the 3-uniform hypergraph 2-coloring problem, we are given a 3-uniform hypergraph $G = (V, E)$ and the goal is to decide if there exists a 2-coloring of the vertices in V such that no edge is monochromatic.

We construct an instance $\Psi' = (G' = (V', E'), p, \mathcal{C}')$ with $\max_{C \in \mathcal{C}'} |C| = 3$ as follows. The vertex set V' consists of two vertices $v^{(0)}, v^{(1)}$ which correspond to the colors we are using. For each vertex $v \in V$, we create an edge e_v in E' with end-points $v^{(0)}, v^{(1)}$. Note that this creates multi-edges between $v^{(0)}$ and $v^{(1)}$ in G'. Intuitively, the orientation of e_v corresponds to the color of vertex v in a valid 2-coloring.

For each 3-uniform hyperedge $e \in E$, say, with endpoints $u, v, w \in V$, we create an SNC constraint $C_e := \{e_u, e_v, e_w\}$ in $\mathcal{C}'$. Intuitively, the SNC constraint requires that not all endpoints of e are assigned to the same vertex, and this models the feasibility of the 2-coloring for G. The following lemma establishes the correctness of the reduction.

▶ **Lemma 2.** *G is 2-colorable if and only if Ψ' is feasible.*

By Lemma 1 and Lemma 2, we obtain the following theorem.

▶ **Theorem 3.** *When $\max(\alpha, \beta) \geq 3$, it is NP-hard to determine the feasibility of (α, β)-instances for the min-max SNC graph balancing problem.*

3 Min-Max $(2, 2)$-SNC Graph Balancing

In this section, we consider the min-max $(2, 2)$-SNC graph balancing problem. First, we present a characterization of feasible instances that can be tested in polynomial-time. Then, we introduce a set of structural properties and modifications on the instance followed with an assignment LP and obtain a 2-approximation for feasible instances of this problem.

Let $\Psi = (G = (V, E),\ p,\ \mathcal{C})$ be an instance of min-max $(2, 2)$-SNC graph balancing, i.e., $|e| \leq 2$ for all $e \in E$ and $|C| = 2$ for all $C \in \mathcal{C}$. To simply the notation, for any $e \in E$ and any $v \in e$, we will use $e \setminus v$ to denote the endpoint of e other than v. Furthermore, $e \setminus v$ is defined to be ϕ if v is the only endpoint of e, i.e., e is a self-loop.

3.1 The Implication Graph and a Feasibility Characterization

In the following, we first define the concept of *implication graph* H for Ψ and a set of *bad implications* in the implication graph H. Then we show that Ψ is feasible if and only if there exists no bad implication in H.

Consider any SNC constraint $\{e, e'\} \in \mathcal{C}$. If v is a common endpoint of e and e', i.e., $v \in e \cap e'$, and if e is already assigned to v, then e' must not be assigned to v in any feasible assignment. In other words, e' must be assigned to $e' \setminus v$. In this scenario, we say that the assignment of e to v *implies* the assignment of e' to $e' \setminus v$.

The above observation defines the directed implication graph $H = (V_H, E_H)$. The vertex set V_H consists of two types of nodes, namely,

- $u_{e,v}$ for each $e \in E$ and each $v \in e$, and
- $u_{e,\phi}$ for each $e \in E$ with $|e| = 1$.

Intuitively, we construct H in a way such that, if $u_{e,v}$ is implied by a directed arc in E_H, then e is supposed to be assigned to v in any feasible assignment. Furthermore, if $u_{e,\phi}$ is implied by an arc, then the instance Ψ is infeasible.

The directed arcs in E_H are defined as follows. For each SNC constraint $\{e, e'\} \in \mathcal{C}$ and each $v \in e \cap e'$, we create two arcs: One from $u_{e,v}$ to $u_{e',e' \setminus v}$ and the other from $u_{e',v}$ to $u_{e,e \setminus v}$. Intuitively, the two arcs indicate that, if one of e or e' is assigned to v, then the other edge must be assigned to the vertex other than v.

Following the above concept, we use $u_{e,v} \xrightarrow{+} u_{e',v'}$ to denote the scenario where there exists a path of nonzero length from $u_{e,v}$ to $u_{e',v'}$ in H. If both $u_{e,v} \xrightarrow{+} u_{e',v'}$ and $u_{e',v'} \xrightarrow{+} u_{e,v}$, then we write $u_{e,v} \xleftrightarrow{+} u_{e',v'}$. Intuitively, if $u_{e,v} \xleftrightarrow{+} u_{e',v'}$, then there exists a cycle that passes both $u_{e,v}$ and $u_{e',v'}$. Furthermore, the assignment of any edge on the nodes of this cycle will uniquely determine the assignments of all the edges on the nodes of the same cycle.

▶ **Definition 4** (Bad Implication)**.** *The following chains of implications are considered* bad*.*

1. *There exists a cycle in H that passes through both $u_{e,v}$ and $u_{e,v'}$ for some $e = \{v, v'\} \in E$, i.e., $u_{e,v} \xleftrightarrow{+} u_{e,v'}$ for some $e = \{v, v'\} \in E$.*
2. *$u_{e,\phi}$ is implied by $u_{e,v}$ for some $e = \{v\} \in E$, i.e., $u_{e,v} \xrightarrow{+} u_{e,\phi}$ for some $e = \{v\} \in E$.*

Clearly, the instance Ψ is infeasible if $u_{e,v}$ and $u_{e,v'}$ imply each other for some $e = \{v, v'\}$ or $u_{e,\phi}$ is implied by $u_{e,v}$ for some $e = \{v\} \in E$. The following lemma, on the contrary, shows that the obvious necessary condition is also sufficient.

▶ **Lemma 5.** *Ψ has a feasible orientation if and only if there is no bad implication in H.*

Although Lemma 5 can be proved directly, we chose to prove it in an implicit way. We show in the following sections that, when there is no bad implication in H, a 2-approximation for Ψ can be computed based on LP-rounding. This completes the proof of Lemma 5.

We also note that, the existence of bad implications can be tested in polynomial-time by simple graph traversal in H. We obtain the following theorem.

▶ **Theorem 6.** *The feasibility of Ψ can be tested in polynomial-time.*

3.2 Unique Edge Orientation and Strongly Connected Components

In the following, we assume that no bad implication exists in the implication graph H. We further simplify the structure of H by identifying

1. edges whose orientations can be uniquely determined, and
2. edges whose orientations are implied by each other.

In the former case, the edges will be assigned directly as *dedicated workloads* that each vertex in V possesses. The latter case corresponds to strongly connected components (SCCs) in H to be contracted and treated as a single vertex. When this process ends, we obtain a simplified implication graph $H'' = (V_{H''}, E_{H''})$, which is directed acyclic, and a dedicated workload function $q\colon V \mapsto \mathbb{Z}^{\geq 0}$ of the vertices. In the following we describe the details.

Unique Edge Orientation

Observe that, the assignment of an edge $e \in E$ with $v \in e$ can be uniquely determined if one of the following two cases holds.

- $e = \{v\}$, i.e., e is a self-loop. Then e must be assigned to v.
- $u_{e,v'} \xrightarrow{+} u_{e,v}$, where $e = \{v, v'\}$. In this case, it also follows that e must be assigned to v.

In addition, provided that the edge e is to be assigned to v, all the nodes (assignments) that are further implied by $u_{e,v}$ in H must be *realized* as well. On the other hand, the opposite direction of the realized assignments, e.g., $u_{e,e\setminus v}$, must never be made and should be removed from the implication graph H.

In the following, we describe a unifying approach to handle the above two cases. We start with a zero dedicated workload function $q \leftarrow 0$ and repeat the following steps while there exists some $v \in e \in E$ such that either $|e| = 1$ or $u_{e,e\setminus v} \xrightarrow{+} u_{e,v}$.

Inside the main while loop, we pick one such $v \in e \in E$ and do the following. Let

$$A \leftarrow \{u_{e,v}\} \cup \left\{ \ell \in V_H \mid u_{e,v} \xrightarrow{+} \ell \right\}$$

be the set of nodes (assignments) in H that are implied by $u_{e,v}$, i.e., the set of nodes reachable from $u_{e,v}$. Intuitively, the assignments in A must be realized as well. On the contrary, let

$$B \leftarrow \left\{ u_{e',e'\setminus v'} \mid u_{e',v'} \in A \right\}$$

be the set of nodes that make the opposite directions of assignments to the nodes in A. Intuitively, the assignments in B must not be realized.

Then, we do the following updates. For each node, say, $u_{e',v'} \in A$, we assign e' to v' and add $p_{e',v'}$ to $q_{v'}$ as dedicated loads of v' accordingly. Then we remove both A and B from H and proceed to the next iteration until there exists no $v \in e \in E$ with $|e| = 1$ or $u_{e,e\setminus v} \xrightarrow{+} u_{e,v}$. In the following, we use Algorithm A to denote the above process.

In the following, we show that, for any $e \in E$ and any $v \in e$, the two nodes $u_{e,v}$ and $u_{e,e\setminus v}$ cannot belong to A at the same time. Hence, the concepts of A and B in Algorithm A are well-defined. We begin with the following structural lemma for H. Intuitively, it provides a reversed symmetric property for the *conjugating pair of nodes* in H in that, whenever $u_{e,v} \xrightarrow{+} u_{e',v'}$ for some $u_{e,v}, u_{e',v'} \in V_H$, their conjugating partners, $u_{e,e\setminus v}$ and $u_{e',e'\setminus v'}$, must have a reversed implication relation $u_{e',e'\setminus v'} \xrightarrow{+} u_{e,e\setminus v}$.

▶ **Lemma 7.** *Let $e, e' \in E$ with $v \in e, v' \in e'$. If $u_{e,v} \xrightarrow{+} u_{e',v'}$, then $u_{e',e'\setminus v'} \xrightarrow{+} u_{e,e\setminus v}$.*

Proof. We prove by induction on the length n of the shortest path from $u_{e,v}$ to $u_{e',v'}$. If $n = 1$, then by the definition of H, we have $\{e, e'\} \in \mathcal{C}$ and $v \in e \cap e'$, and $v = e' \setminus v'$. Hence, when e' is assigned to v, e must be assigned to $e \setminus v$. Therefore we have $u_{e',e'\setminus v'} \xrightarrow{+} u_{e,e\setminus v}$.

Assume that the statement holds when the length of the shortest path from $u_{e,v}$ to $u_{e',v'}$ is at most n. Then for the length $n+1$, pick an arbitrary intermediate vertex ℓ on the shortest path from $u_{e,v}$ to $u_{e',v'}$. That is to say, $u_{e,v} \xrightarrow{+} \ell$ and $\ell \xrightarrow{+} u_{e',v'}$. It follows that the lengths of both subpaths is at most n. So by assumption, we have $u_{e',e'\setminus v'} \xrightarrow{+} \ell' \xrightarrow{+} u_{e\setminus v}$, where ℓ' is the conjugating pair of ℓ. This proves the lemma. ◀

The following lemma shows that the concepts of A and B in Algorithm A are well-defined.

▶ **Lemma 8.** *For any $v \in e \in E$, $u_{e,v} \in A$ implies that $u_{e,e\setminus v} \notin A$.*

Proof. Consider any iteration in Algorithm A. Let (v^*, e^*), where $v^* \in e^* \in E$, denote the pair that is selected in the beginning of the iteration such that either $|e^*| = 1$ or $u_{e^*,e^*\setminus v^*} \xrightarrow{+} u_{e^*,v^*}$.

Assume for contradiction that, for some $v \in e \in E$, both $u_{e,v}$ and $u_{e,e\setminus v}$ are in A. Depending on whether or not $e = e^*$, we distinguish two cases and show that they both lead to bad implications in H. Note that this will be a contradiction to our assumption in H.

- $e = e^*$ and $v = v^*$, i.e., (e, v) is the pair chosen in the beginning of this iteration. In this case, since $u_{e^*,e^*\setminus v^*} = u_{e,e\setminus v} \in A$, we have $u_{e^*,v^*} \xrightarrow{+} u_{e^*,e^*\setminus v^*}$, which is a bad implication.
- Assume that $e \neq e^*$. Since both $u_{e,v}, u_{e,e\setminus v} \in A$, it follows that

$$u_{e^*,v^*} \xrightarrow{+} u_{e,v} \quad \text{and} \quad u_{e^*,v^*} \xrightarrow{+} u_{e,e\setminus v} \tag{1}$$

hold at the same time. By Lemma 7, this implies that

$$u_{e,e\setminus v} \xrightarrow{+} u_{e^*,e^*\setminus v^*} \quad \text{and} \quad u_{e,v} \xrightarrow{+} u_{e^*,e^*\setminus v^*} \tag{2}$$

hold at the same time. We further consider the two subcases for which $|e^*| = 1$ or not.

- If $|e^*| \neq 1$, then we have $u_{e^*,e^*\setminus v^*} \xrightarrow{+} u_{e^*,v^*}$ by the condition we pick at the beginning of the while loop. Then we have $u_{e,e\setminus v} \xrightarrow{+} u_{e^*,e^*\setminus v^*} \xrightarrow{+} u_{e^*,v^*} \xrightarrow{+} u_{e,v}$ by (1) and (2), which is bad.
- If $|e^*| = 1$, then $u_{e^*,v^*} \xrightarrow{+} u_{e,v} \xrightarrow{+} u_{e^*,e^*\setminus v^*}$ is a bad implication since $u_{e^*,e^*\setminus v^*} = u_{e^*,\phi}$.

In all cases, it leads to a bad implication, which is a contradiction to the assumption that Ψ is a feasible instance. This proves the lemma. ◀

By Algorithm A, we assume in the following that there are no self-loops in G and for any $e = \{v, v'\} \in E$, none of $u_{e,v}$ or $u_{e,v'}$ imply each other. Furthermore, we have a dedicated workload function q for the vertices in V.

Handling the Strongly Connected Components

Consider the case that $\ell \overset{+}{\Leftrightarrow} \ell'$ for some $\ell, \ell' \in V_H$. Clearly this corresponds to a directed cycle of implications, say, C, in H and constitutes as part of a strongly connected component (SCC), say, C'. It follows that, for any node on the cycle, say $u_{e,v} \in C$, if the orientation of e is determined, then the orientation of all the remaining edges to which the nodes on the cycle correspond is also determined.

In fact, it is straightforward to verify that, the orientation of all the edges in the component C' are mutually bound to each other. From this observation, the whole component C' can be treated as a single node in the implication graph H, since the assignments of all the edges on the nodes of this component are bound together.

In the following we formally define this concept. Let H' be the updated implication graph after Algorithm A is applied and $C'_1, C'_2, \ldots, C'_k$ be the SCCs we have in H'.

Define the contracted implication graph $H'' = (V_{H''}, E_{H''})$ as follows. For each $1 \leq i \leq k$, we have a vertex v_i in $V_{H''}$ that represents the component C'_i. For any $1 \leq i, j \leq k$, we draw an arc (v_i, v_j) in $E_{H''}$ if there is an arc (ℓ, ℓ') that connects some $\ell \in C'_i$ to some $\ell' \in C'_j$.

Intuitively, the graph H'' is obtained by contracting each SCC in H' into a single vertex. Since there is a one-to-one correspondence between SCCs in H' and the vertices in $V_{H''}$, we will use $\delta(s)$ for any $s \in V_{H''}$ to denote the SCC to which s corresponds in H'. The following lemma is straightforward to verify.

▶ **Lemma 9.** *H'' is acyclic.*

The following structural lemma for SCCs in H', obtained from Lemma 7, shows that SCCs in H' also form conjugating pairs, regardless of their sizes.

▶ **Lemma 10.** *For any $e, e' \in E$ with $v \in e$, $v' \in e'$, if $u_{e,v}$ and $u_{e',v'}$ belong to the same SCC, then $u_{e,e\setminus v}$ and $u_{e',e'\setminus v'}$ must belong to the same SCC as well.*

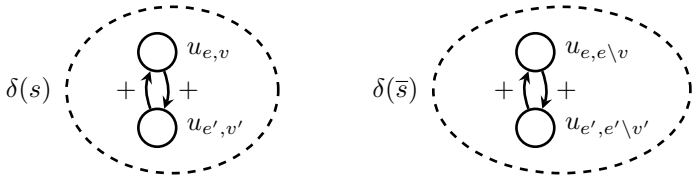


Figure 1 An illustration of the definition of conjugating pairs in $V_{H''}$.

Lemma 10 allows the concept of conjugation for SCCs to be defined. Formally, for any $s \in V_{H''}$ and any $u_{e,v} \in \delta(s)$, define $\overline{s}$ to be the vertex in $V_{H''}$ such that $\delta(\overline{s})$ contains the node $u_{e,e\setminus v}$. Note that, by Lemma 10, the vertex $\overline{s}$ is uniquely defined for each $s \in V_{H''}$. Also see Figure 1 for an illustration.

3.3 A 2-Approximation Algorithm

Let $H'' = (V_{H''}, E_{H''})$ be the simplified implication graph we obtained from Section 3.2. Note that H'' is acyclic by Lemma 9. Now we are ready to describe our assignment LP LP-(T) for this problem and our 2-approximation algorithm.

For each vertex $s \in V_{H''}$, we introduce a decision variable $x_s \in \{0, 1\}$ to indicate whether or not the assignments specified in the nodes of the SCC $\delta(s)$ should be realized. In this regard, for any $v \in V$, define $p_s(v) := \sum_{e \in E \text{ s.t. } u_{e,v} \in \delta(s)} p_{e,v}$ to be the workload vertex $v \in V$ will receive, if the assignments in $\delta(s)$ are realized.

Let $T \geq 0$ be the target maximum workload of the vertices to be achieved. We have the following feasibility LP relaxation with respect to the target value T.

$$\sum_{s \in V_{H''}} p_s(v) \cdot x_s + q_v \leq T, \qquad \forall v \in V, \tag{3a}$$

$$x_s + x_{\overline{s}} = 1, \qquad \forall s \in V_{H''}, \tag{3b}$$

$$x_s \leq x_{s'}, \qquad \forall (s, s') \in E_{H''}, \tag{3c}$$

$$x_s \geq 0, \qquad \forall s \in V_{H''}. \tag{3d}$$

In the above LP formulation, the constraint (3a) models the maximum workload T for each $v \in V$. The second constraint (3b) states that, for each conjugating pair of SCCs, exactly one type of orientation is made. The third constraint (3c) models the arc of implication in H'', namely, if $(s, s') \in E_{H''}$ and x_s is 1, then $x_{s'}$ must also be 1.

The Algorithm

Our algorithm goes as follows. First, it uses binary search to compute the smallest T_0 such that LP-(T_0) is feasible. Let $\hat{\sigma}$ be an optimal assignment for Ψ and $\hat{T}$ be the maximum workload of $\hat{\sigma}$. Then, it follows that T_0 must be a lower-bound of $\hat{T}$, since $\hat{\sigma}$ corresponds to a set of feasible solution for LP-($\hat{T}$). Let x^* be a fractional solution for LP-(T_0).

In the following, we describe a procedure that rounds x^* into an integer solution $\tilde{x}$ such that the workload of each vertex is at most doubled. Define

$$S^{\neq} := \left\{ s \in V_{H''} \mid x^*_s \neq \frac{1}{2} \right\} \quad \text{and} \quad S^{=} := \left\{ s \in V_{H''} \mid x^*_s = \frac{1}{2} \right\}.$$

For any $s \in S^{\neq}$, define

$$\tilde{x}_s := \begin{cases} 1, & \text{if } x^*_s > 1/2, \\ 0, & \text{if } x^*_s < 1/2. \end{cases}$$

By constraint (3b), if $x^*_s > 1/2$ for some $s \in S^{\neq}$, then it follows that $x^*_{\overline{s}} < 1/2$ and vice versa. Hence, the above setting of $\tilde{x}$ keeps constraint (3b) satisfied. Furthermore, the workload each vertex receives is at most doubled since x^*_s is rounded up only when it is at least 1/2.

However, for any component $s \in S^{=}$, we have $\overline{s} \in S^{=}$ as well. Hence, x^*_s and $x^*_{\overline{s}}$ cannot both be rounded up at the same time since constraint (3b) will be violated. To resolve the rounding problem for components in $S^{=}$, we use the fact that $H'' \setminus S^{\neq}$ is still a DAG and consider the topological order of the components in $S^{=}$.

Let $S := H'' \setminus S^{\neq}$. Repeat the following steps until S becomes empty. In each iteration, pick a component $s \in S$ with zero out-degree. Intuitively, the orientation of s does not affect the orientation of the remaining components in S. We set $\tilde{x}_s$ to be 1 and $\tilde{x}_{\overline{s}}$ to be zero. Then we remove both s and $\overline{s}$ from S. This process is repeated until S becomes empty.

To obtain an orientation for the edges in E, we make the assignments specified in each SCC s with $\tilde{x}_s = 1$. In particular, for each $s \in V_{H''}$ with $\tilde{x}_s = 1$ and each node, say, $u_{e,v} \in \delta(s)$, we assign e to v by setting $\sigma(e) = v$. Then we output σ to be the approximate solution for Ψ.

The following lemma shows that, in any iteration of the above rounding procedure, if a component s has zero out-degree, then $\overline{s}$ must have a zero in-degree. This shows that our rounding procedure for components in $S^{=}$ is well-defined.

▶ **Lemma 11.** *For any $s \in V_{H''}$, if s has zero out-degree, then $\overline{s}$ must have a zero in-degree.*

The following lemma shows that σ is a feasible orientation for Ψ. Note that this also completes the proof for Lemma 5 and our characterization on the feasibility of $(2,2)$-SNC graph balancing.

▶ **Lemma 12.** *σ is feasible for Ψ.*

Proof. As an integer solution for LP-(T) corresponds naturally to a feasible assignment, it suffices to show that $\tilde{x}$ is feasible for LP-(T) for some T.

Clearly $\tilde{x}$ satisfies constraint (3b) and (3d) in LP-(T_0). For the constraint (3c), consider any $(s, s') \in E_{H''}$. Since x^* is a feasible solution for LP-(T_0), we have $x^*_s \leq x^*_{s'}$. We will show that $\tilde{x}_s \leq \tilde{x}_{s'}$. Depending on the values of x^*_s and $x^*_{s'}$, we consider the following cases.

- If $1/2 < x^*_s \leq x^*_{s'}$ or $x^*_s \leq x^*_{s'} < 1/2$, then $\tilde{x}_s = \tilde{x}_{s'}$ by our rounding scheme.
- If $x^*_s < 1/2$ or $1/2 < x^*_{s'}$, then $\tilde{x}_s = 0$ for the former case or $\tilde{x}_{s'} = 1$ for the latter case. In both cases, $\tilde{x}_s \leq \tilde{x}_{s'}$ holds.
- For the remaining case for which $x^*_s = x^*_{s'} = 1/2$, assume for contradiction that constraint (3c) is not satisfied, i.e., $\tilde{x}_s = 1$ and $\tilde{x}_{s'} = 0$.
  Since $\tilde{x}_s = 1$, we know that s' has already been removed from H'' when s is selected to be rounded up by the algorithm. Since $\tilde{x}_{s'} = 0$, we know that s' was removed because its conjugating pair was selected and removed. But this will be a contradiction to Lemma 11 since the in-degree of s' was at least 1 at that time. Hence, constraint (3c) also holds.

This proves the feasibility of $\tilde{x}$ for LP-(T) for some T. ◀

It remains to prove the following theorem.

▶ **Theorem 13.** *σ can be computed in polynomial-time and is a 2-approximation for Ψ.*

Proof. It is clear that the computation can be done in polynomial-time. For each vertex $v \in V$, we know that the workload of v is

$$\sum_{e \in \sigma^{-1}(v)} p_{e,v} \;=\; \sum_{s \in V_{H''}} \tilde{x}_s \cdot \left(\sum_{u_{e,v} \in \delta(s)} p_{e,v} \right) \;+\; q_v \;=\; \sum_{s \in V_{H''}} \tilde{x}_s \, p_s(v) \;+\; q_v \,.$$

Observe that for any $s \in V_{H''}$, $\tilde{x}_s = 1$ only when $x^*_s \geq 1/2$. Hence we have $\tilde{x}_s \leq 2 \cdot x^*_s$. It follows that, for each vertex $v \in V$, we have

$$\sum_{s \in V_{H''}} \tilde{x}_s \, p_s(v) \;+\; q_v \;\leq\; \sum_{s \in V_{H''}} 2 \cdot x^*_s \, p_s(v) \;+\; q_v \;\leq\; 2 \cdot T_0 \;\leq\; 2 \cdot \hat{T},$$

where $\hat{T}$ is the maximum workload of the optimal assignment $\hat{\sigma}$ and in the last inequality we use the fact that T_0 is the smallest value such that LP-(T_0) is feasible. ◀

Integrality Gap of LP-(T)

In the following we show that the integrality gap of LP-(T) is 2. This shows that the approximation ratio we obtained for this problem is tight in terms of the LP we use. Consider the instance shown in Figure 2 with the weights $p_{e_1,a} = p_{e_5,b} = p_{e_3,a} = p_{e_3,b} = 1$ and all other 0, and the SNC constraints $\mathcal{C} = \{\{e_1, e_2\}, \{e_2, e_3\}, \{e_3, e_4\}, \{e_4, e_5\}\}$.

Observe that no matter e_3 is oriented to a or b, it always forces e_1 or e_5 to be oriented to the same vertex to which e_3 is oriented. Hence, the maximum workload of any feasible orientation is at least 2. On the other hand, consider the simplified implication graph H'' of the instance, shown on the r.h.s. of Figure 2. Observe that, by setting $x_s = 1/2$ for all $s \in V_{H''}$, all the constraints of LP-(T) with $T = 1$ are satisfied and we obtain a fractional orientation with maximum workload 1. This shows that the integrality gap is at least 2.

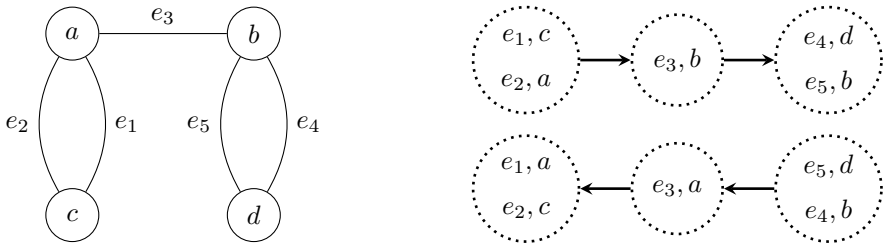


Figure 2 An example which shows that the integrality gap of LP-(T) is 2. On the right hand side, we use e, v to denote $u_{e,v}$ for notational simplicity.

Extension to Weighted Completion Time with SNC Constraints.

Our approach for graph balancing with SNC constraints is general and can be applied to similar settings to obtain similar approximation guarantees. In the following, we sketch how our algorithm framework can be used to obtain a 2-approximation when the objective is to minimize the weighted completion time, instead of maximum workload.

In fact, apart from the different objective function we need in the LP formulation, the remaining parts are exactly the same. We have the following corollary.

▶ **Corollary 14.** *We can compute a* 2*-approximation for the* $(2, 2)$*-SNC graph balancing problem to minimize the weighted completion time.*

4 Min-Max $(2, 2)$-SNC Graph Balancing on Restricted Graphs

In this section, we present both approximation and hardness results for min-max $(2, 2)$-SNC graph balancing on restricted graphs to describe a complete landscape of this problem.

Let $\Psi = (G = (V, E),\ p,\ \mathcal{C})$ be an instance of min-max $(2, 2)$-SNC graph balancing.

▶ **Theorem 15.** *There is an FPTAS for min-max* $(2, 2)$*-SNC graph balancing when* $|V| = 2$*. When* $|V| \geq 3$*, this problem is at least as hard as vertex cover to approximate.*

Note that, Theorem 15 provides a clear landscape on this problem and shows that, assuming the unique game conjecture (UGC), the approximation guarantee we obtained in this work is already tight even when $|V| = 3$.

References

1 Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. Combinatorial algorithm for restricted max-min fair allocation. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1357–1372. SIAM, 2015. doi:10.1137/1.9781611973730.90.

2 Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. In Ashish Goel, Klaus Jansen, José D. P. Rolim, and Ronitt Rubinfeld, editors, *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Boston, MA, USA, August 25-27, 2008. Proceedings*, volume 5171 of *Lecture Notes in Computer Science*, pages 10–20. Springer, 2008. doi:10.1007/978-3-540-85363-3_2.

3 Nikhil Bansal, Aravind Srinivasan, and Ola Svensson. Lift-and-round to improve weighted completion time on unrelated machines. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 156–167. ACM, 2016. doi:10.1145/2897518.2897572.

4 Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 31–40. ACM, 2006. doi:10.1145/1132516.1132522.

5 Alok Baveja, Xiaoran Qu, and Aravind Srinivasan. Approximating weighted completion time via stronger negative correlation. *Journal of Scheduling*, pages 1–10, 2023.

6 Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *SIGecom Exch.*, 5(3):11–18, 2005. doi:10.1145/1120680.1120683.

7 Hans L. Bodlaender, Klaus Jansen, and Gerhard J. Woeginger. Scheduling with incompatible jobs. In Ernst W. Mayr, editor, *Graph-Theoretic Concepts in Computer Science, 18th International Workshop, WG '92, Wiesbaden-Naurod, Germany, June 19-20, 1992, Proceedings*, volume 657 of *Lecture Notes in Computer Science*, pages 37–49. Springer, 1992. doi:10.1007/3-540-56402-0_34.

8 Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 107–116. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.51.

9 Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. On (1,)-restricted assignment makespan minimization. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1087–1101. SIAM, 2015. doi:10.1137/1.9781611973730.73.

10 Deeparnab Chakrabarty and Kirankumar Shiragur. Graph balancing with two edge types. *CoRR*, abs/1604.06918, 2016. arXiv:1604.06918.

11 Siu-Wing Cheng and Yuchen Mao. Restricted max-min fair allocation. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 37:1–37:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.37.

12 Siu-Wing Cheng and Yuchen Mao. Restricted max-min allocation: Approximation and integrality gap. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 38:1–38:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.38.

13 Syamantak Das and Andreas Wiese. On minimizing the makespan when some jobs cannot be assigned on the same machine. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.31.

14 Sami Davies, Thomas Rothvoss, and Yihao Zhang. A tale of santa claus, hypergraphs and matroids. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2748–2757. SIAM, 2020. doi:10.1137/1.9781611975994.167.

15 Irit Dinur, Oded Regev, and Clifford D. Smyth. The hardness of 3 - uniform hypergraph coloring. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, page 33. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181880.

16 Tomás Ebenlendr, Marek Krcál, and Jirí Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 483–490. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347135.

17 Uriel Feige. On allocations that maximize fairness. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 287–293. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347114.

18 Penny Haxell and Tibor Szabó. Improved integrality gap in max-min allocation: or topology at the north pole. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2875–2897. SIAM, 2023. doi:10.1137/1.9781611977554.ch109.

19 Chien-Chung Huang and Sebastian Ott. A combinatorial approximation algorithm for graph balancing with light hyper edges. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 49:1–49:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.49.

20 Klaus Jansen, Alexandra Lassota, and Marten Maack. Approximation algorithms for scheduling with class constraints. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 349–357. ACM, 2020. doi:10.1145/3350755.3400247.

21 Klaus Jansen and Lars Rohwedder. On the configuration-lp of the restricted assignment problem. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2670–2678. SIAM, 2017. doi:10.1137/1.9781611974782.176.

22 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 217–224. IEEE Computer Society, 1987. doi:10.1109/SFCS.1987.8.

23 Daniel R. Page and Roberto Solis-Oba. A 3/2-approximation algorithm for the graph balancing problem with two weights. *Algorithms*, 9(2):38, 2016. doi:10.3390/a9020038.

24 Daniel R. Page and Roberto Solis-Oba. Makespan minimization on unrelated parallel machines with a few bags. In Shaojie Tang, Ding-Zhu Du, David L. Woodruff, and Sergiy Butenko, editors, *Algorithmic Aspects in Information and Management - 12th International Conference, AAIM 2018, Dallas, TX, USA, December 3-4, 2018, Proceedings*, volume 11343 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2018. doi:10.1007/978-3-030-04618-7_3.

25 Tytus Pikies, Krzysztof Turowski, and Marek Kubale. Scheduling with complete multipartite incompatibility graph on parallel machines: Complexity and algorithms. *Artif. Intell.*, 309:103711, 2022. doi:10.1016/j.artint.2022.103711.

26 Petra Schuurman and Gerhard J Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999.

27 Ola Svensson. Santa claus schedules jobs on unrelated machines. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 617–626. ACM, 2011. doi:10.1145/1993636.1993718.

28 José Verschae and Andreas Wiese. On the configuration-lp for scheduling on unrelated machines. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 530–542. Springer, 2011. doi:10.1007/978-3-642-23719-5_45.

29 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

On the Line-Separable Unit-Disk Coverage and Related Problems

Gang Liu
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Haitao Wang
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Abstract

Given a set P of n points and a set S of m disks in the plane, the disk coverage problem asks for a smallest subset of disks that together cover all points of P. The problem is NP-hard. In this paper, we consider a line-separable unit-disk version of the problem where all disks have the same radius and their centers are separated from the points of P by a line ℓ. We present an $m^{2/3}n^{2/3}2^{O(\log^*(m+n))} + O((n+m)\log(n+m))$ time algorithm for the problem. This improves the previously best result of $O(nm + n\log n)$ time. Our techniques also solve the line-constrained version of the problem, where centers of all disks of S are located on a line ℓ while points of P can be anywhere in the plane. Our algorithm runs in $O(m\sqrt{n} + (n+m)\log(n+m))$ time, which improves the previously best result of $O(nm\log(m+n))$ time. In addition, our results lead to an algorithm of $n^{10/3}2^{O(\log^* n)}$ time for a half-plane coverage problem (given n half-planes and n points, find a smallest subset of half-planes covering all points); this improves the previously best algorithm of $O(n^4\log n)$ time. Further, if all half-planes are lower ones, our algorithm runs in $n^{4/3}2^{O(\log^* n)}$ time while the previously best algorithm takes $O(n^2\log n)$ time.

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

Keywords and phrases disk coverage, line-separable, unit-disk, line-constrained, half-planes

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.51

Related Version *Full Version*: `http://arxiv.org/abs/2309.03162`

Funding This research was supported in part by NSF under Grants CCF-2005323 and CCF-2300356.

1 Introduction

Given a set P of n points and a set S of m disks in the plane, the *disk coverage* problem asks for a smallest subset of disks such that every point of P is covered by at least one disk in the subset. The problem is NP-hard, even if all disks have the same radius [15, 20]. Polynomial time approximation algorithms have been proposed for the problem and many of its variants, e.g., [1, 6, 8, 9, 16, 19].

Polynomial time exact algorithms are known for certain special cases. If all points of P are inside a strip bounded by two parallel lines and the centers of all disks lie outside the strip, then the problem is solvable in polynomial time [3]. If all disks of S contain the same point, polynomial time algorithms also exist [12, 13]; in particular, applying the result in [8] (i.e., Corollary 1.7) yields an $O(mn^2(m+n))$ time algorithm. In order to devise an efficient approximation algorithm for the general coverage problem (without any constraints), the *line-separable* version was considered in the literature [3, 7, 11], where disk centers are separated from the points by a given line ℓ. A polynomial time 4-approximation algorithm is given in [7]. Ambühl et al. [3] derived an exact algorithm of $O(m^2n)$ time. An improved $O(nm + n\log n)$ time algorithm is presented in [11] and another algorithm in [21] runs in $O(n\log n + m^2\log n)$ in the worst case.

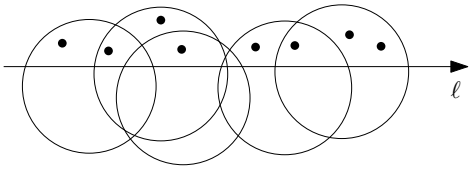


Figure 1 Illustrating the line-separable unit-disk case.

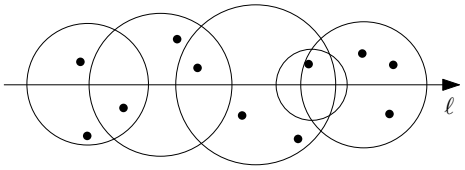


Figure 2 Illustrating the line-constrained case (all disks are centred on ℓ).

The *line-constrained* version of the disk coverage problem has also been studied, where disk centers are on the x-axis while points of P can be anywhere in the plane. Pedersen and Wang [21] considered the weighted case in which each disk has a weight and the objective is to minimize the total weight of the disks in the subset that cover all points. Their algorithm runs in $O((m+n)\log(m+n)+\kappa\log m)$ time, where κ is the number of pairs of disks that intersect and $\kappa = O(m^2)$ in the worst case. They reduced the runtime to $O((m+n)\log(m+n))$ for the *unit-disk case*, where all disks have the same radius, as well as the L_∞ and L_1 cases, where the disks are squares and diamonds, respectively [21]. The 1D problem where disks become segments on a line and points are on the same line is also solvable in $O((m+n)\log(m+n))$ [21]. Other types of line-constrained coverage problems have also been studied in the literature, e.g., [2, 4, 5, 18].

A related problem is when disks of S are half-planes. For the weighted case, Chan and Grant [8] proposed an algorithm for the lower-only case where all half-planes are lower ones; their algorithm runs in $O(n^4)$ time when $m = n$. With the observation that a half-plane may be considered as a unit disk of infinite radius, the techniques of [21] solve the problem in $O(n^2 \log n)$ time. For the general case where both upper and lower half-planes are present, Har-Peled and Lee [17] solved the problem in $O(n^5)$ time. Pedersen and Wang [21] showed that the problem can be reduced to $O(n^2)$ instances of the lower-only case problem and thus can be solved in $O(n^4 \log n)$ time. To the best of our knowledge, we are not aware of any previous work particularly on the unweighted half-plane coverage problem.

1.1 Our result

We assume that ℓ is the x-axis and all disk centers are below or on ℓ while all points of P are above or on ℓ. We consider the line-separable version of the disk coverage problem with the following *single-intersection condition*: For any two disks, their boundaries intersect at most once in the half-plane above ℓ. Note that this condition is satisfied in both the unit-disk case (see Fig 1) and the line-constrained case (see Fig. 2; more to explain below). Hence, an algorithm for this line-separable single-intersection case works for both the unit-disk case and the line-constrained case. Note that all problems considered in this paper are unweighted case in the L_2 metric.

For the above line-separable single-intersection problem, we give an algorithm of $O(m\sqrt{n}+(n+m)\log(n+m))$ time in Section 3. Based on observations, we find that some disks are "useless" and thus can be pruned from S. After pruning those useless disks, the remaining disks have certain property so that we can reduce the problem to the 1D problem, which can then be easily solved. The overall algorithm is fairly simple conceptually. One challenge, however, is to show the correctness, namely, to prove why those "useless" disks are indeed useless. The proof is rather lengthy and technical. The bottleneck of the algorithm is to find those useless disks, for which we utilize the cuttings [10].

The line-constrained problem. Observe that the line-constrained problem where all disks of S are centered on a line ℓ while points of P can be anywhere in the plane is also a special case of the line-separable single-intersection problem. Indeed, for each point p of P below ℓ, we could replace p by its symmetric point with respect to ℓ; in this way, we can obtain a set of points that are all above ℓ. It is not difficult to see that an optimal solution using this new set of points is also an optimal solution for P. Further, since disks are centered on ℓ, although their radii may not be equal, boundaries of any two disks intersect at most once above ℓ. Hence, the problem is an instance of the line-separable single-intersection case. As such, applying our algorithm in Section 3 solves the line-constrained problem in $O(m\sqrt{n}+(n+m)\log(n+m))$ time; this improves the previous algorithm in [21], which runs in $O(n\log n+m^2\log m)$ time in the worst case.

The unit-disk case. To solve the line-separable unit-disk case, the algorithm in Section 3 still works. However, by making use of the property that all disks have the same radius, we further improve the runtime to $m^{2/3}n^{2/3}2^{O(\log^*(m+n))}+O((m+n)\log(m+n))$ in Section 4. This improves the $O(nm+n\log n)$ time algorithm in [11] as well as the $O(n\log n+m^2\log n)$ time one in [21]. The main idea of the improvement (over the algorithm in Section 3) is to explore the duality of certain subproblems in the algorithm (i.e., consider the corresponding problems on the centers of all unit disks of S and the unit disks centered at the points of P). We derive new algorithms for these dual subproblems and then combine them with the algorithms in Section 3 using recursion (the number of recursions is $O(\log^*(n+m))$ and this is why there is a factor $2^{O(\log^*(m+n))}$ in the time complexity).

The half-plane coverage problem. As in [21], our techniques also solve the half-plane coverage problem. Specifically, for the lower-only case, let ℓ be a horizontal line that is below all points of P. If we consider each half-plane as a unit disk of infinite radius with center below ℓ, then the problem becomes an instance of the line-separable unit-disk coverage problem. Therefore, applying our result leads to an $m^{2/3}n^{2/3}2^{O(\log^*(m+n))}+O((m+n)\log(m+n))$ time algorithm. When $m=n$, this is $n^{4/3}2^{O(\log^* n)}$ time, improving the previous algorithm of $O(n^2\log n)$ time [21]. For the general case where both the upper and lower half-plane are present, using the method in [21] that reduces the problem to $O(n^2)$ instances of the lower-only case, the problem is now solvable in $m^{2/3}n^{8/3}2^{\log^*(m+n)}+O(n^2(m+n)\log(m+n))$ time. When $m=n$, this is $n^{10/3}2^{O(\log^* n)}$ time, improving the previous algorithm of $O(n^4\log n)$ time [21].

2 Preliminaries

This section introduces some concepts and notations that we will use in the rest of the paper.

We follow the notation defined in Section 1, e.g., P, S, m, n, ℓ. Without loss of generality, we assume that ℓ is the x-axis and points of P are all above or on ℓ while centers of disks of S are all below or on ℓ. Under this setting, for each disk $s\in S$, only its portion above ℓ matters for our coverage problem. Hence, unless otherwise stated, a disk s only refers to its portion above ℓ. As such, the boundary of s consists of an *upper arc*, i.e., the boundary arc of the original disk above ℓ, and a *lower segment*, i.e., the intersection of s with ℓ. Notice that s has a single leftmost (resp., rightmost) point, which is the left (resp., right) endpoint of the lower segment of s.

We assume that each point of P is covered by at least one disk since otherwise there would be no feasible solution. Our algorithm is able to check whether the assumption is met. We make a general position assumption that no point of P lies on the boundary of a disk

and no two points of A have the same x-coordinate, where A is the union of P and the set of the leftmost and rightmost points of all disks. Degenerated cases can be easily handled by standard techniques of perturbation, e.g., [14].

For any point p in the plane, we denote its x- and y-coordinates by $x(p)$ and $y(p)$, respectively. We sort all the points of P in ascending order of their x-coordinates, resulting in a sorted list $p_1, p_2, \cdots, p_n$. We also sort all disks in ascending order of the x-coordinates of their leftmost points, resulting in a sorted list $s_1, s_2, \cdots, s_m$. We use $S[i, j]$ to denote the subset $\{s_i, s_{i+1}, \cdots, s_j\}$; for convenience, $S[i, j] = \emptyset$ if $i > j$. For each disk s_i, let l_i and r_i denote its leftmost and rightmost points, respectively.

For any disk s, we use $S_l(s)$ (resp., $S_r(s)$) to denote the set of disks S whose leftmost points are to the left (resp., right) of that of s. As such, if the index of s is i, then $S_l(s) = S[1, i-1]$ and $S_r(s) = S[i+1, m]$. If disk $s' \in S_l(s)$, then we also say that s' is *to the left* of s; similarly, if $s' \in S_r(s)$, then s' is *to the right* of s.

For a point $p_i \in P$ and a disk $s_k \in S$, we say that p_i is *vertically above* s_k if p_i is outside s_k and $x(l_k) < x(p_i) < x(r_k)$.

If S' is a subset of S that form a coverage of P, then we call S' a *feasible solution*. If S' is a feasible solution of minimum size, then S' is an *optimal solution*.

The non-containment property. Suppose a disk s_i contains another disk s_j. Then s_j is redundant for our problem since any point covered by s_j is also covered by s_i. Those redundant disks can be easily identified and removed from S in $O(m \log m)$ time (indeed, this is a 1D problem by observing that s_i contains s_j if and only if the lower segment of s_i contains that of s_j). Hence, for solving our problem, we first remove such redundant disks and work on the remaining disks. For simplicity, from now on we assume that no disk of S contains another. Therefore, S has the following *non-containment* property, which our algorithm relies on.

▶ **Observation 1.** (Non-Containment Property) *For any two disks $s_i, s_j \in S$, $x(l_i) < x(l_j)$ if and only if $x(r_i) < x(r_j)$.*

Cuttings. One algorithmic tool we use is the cuttings [10]. Let H denote the set of the upper arcs of all disks of S. Note that $|H| = m$.

For a parameter r with $1 \leq r \leq m$, a $(1/r)$-*cutting* Ξ of size $O(r^2)$ for H is a collection of $O(r^2)$ constant-complexity cells whose union covers the plane such that for any cell σ, $|H_\sigma| \leq m/r$ holds, where H_σ is the subset of arcs of H that intersect the interior of σ (H_σ is often called the *conflict list* in the literature). In our algorithm descriptions, we often use S_σ, defined as the subset of disks whose upper arcs are in H_σ.

Our algorithm actually uses *hierarchical cuttings* [10]. A cutting Ξ' *c-refines* a cutting Ξ if each cell of Ξ' is contained in a single cell of Ξ and every cell of Ξ contains at most c cells of Ξ'. Let Ξ_0 denote the cutting whose single cell is the entire plane. We define cuttings $\{\Xi_0, \Xi_1, ..., \Xi_k\}$, in which each Ξ_i, $1 \leq i \leq k$, is a $(1/\rho^i)$-cutting of size $O(\rho^{2i})$ that c-refines Ξ_{i-1}, for two constants ρ and c. By setting $k = \lceil \log_\rho r \rceil$, the last cutting Ξ_k is a $(1/r)$-cutting. The sequence $\{\Xi_0, \Xi_1, ..., \Xi_k\}$ of cuttings is called a *hierarchical* $(1/r)$-*cutting* of H. For a cell σ' of Ξ_{i-1}, $1 \leq i \leq k$, that fully contains cell σ of Ξ_i, we say that σ' is the *parent* of σ and σ is a *child* of σ'. Thus the hierarchical $(1/r)$-cutting can be viewed as a tree structure with Ξ_0 as the root. We often use Ξ to denote the set of all cells in all cuttings Ξ_i, $0 \leq i \leq k$.

A hierarchical $(1/r)$-cutting of H can be computed in $O(mr)$ time, e.g., by the algorithm in [22], which adapts Chazelle's algorithm [10] for hyperplanes. The algorithm also produces the conflict lists H_σ (and thus S_σ) for all cells $\sigma \in \Xi$, implying that the total size of these

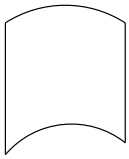


Figure 3 Illustrating a pseudo-trapezoid.

conflict lists is bounded by $O(mr)$. In particular, each cell of the cutting produced by the algorithm of [22] is a (possibly unbounded) *pseudo-trapezoid* that typically has two vertical line segments as left and right sides, a sub-arc of an arc of H as a top side (resp., bottom side) (see Fig. 3).

3 The line-separable single-intersection case

In this section, we present our algorithm for the disk coverage problem in the line-separable single-intersection case. We follow the notation defined in Section 2.

For each disk $s_i \in S$, we define two indices $a(i)$ and $b(i)$ of points of P (where $p_{a(i)}$ and $p_{b(i)}$ are not contained in s_i), which are critical to our algorithm.

▶ **Definition 2.**

- *Among all points of P covered by the union of the disks of $S[1, i-1]$ but not covered by s_i, define $a(i)$ to be the largest index of these points; if no such point exists, then let $a(i) = 0$.*
- *Among all points of P covered by the union of the disks of $S[i+1, m]$ but not covered by s_i, define $b(i)$ to be the smallest index of these points; if no such point exists, then let $b(i) = n + 1$.*

We now describe our algorithm. Although the algorithm description looks simple, it is quite challenging to prove the correctness. Due to the space limit, the correctness proof is in the full version of the paper. The algorithm implementation, which is also not trivial, is presented in Section 3.1.

Algorithm description. The algorithm has three main steps.

1. We first compute $a(i)$ and $b(i)$ for all disks $s_i \in S$. We will show in Section 3.1 that this can be done in $O(m\sqrt{n} + (n+m)\log(n+m))$ time using cuttings.
2. For each disk s_i, if $a(i) \geq b(i)$, we say that s_i a *prunable disk*. Let S^* denote the subset of disks of S that are not prunable. We prove in the full version of this paper that S^* contains an optimal solution for the coverage problem on P and S. This means that it suffices to work on S^* and P.
3. We reduce the disk coverage problem on S^* and P to a 1D coverage problem as follows. For each point of P, we project it vertically onto ℓ. Let P' be the set of all projected points. For each disk $s_i \in S^*$, we create a line segment on ℓ whose left endpoint has x-coordinate equal to $x(p_{a(i)+1})$ and whose right endpoint has x-coordinate equal to $x(p_{b(i)-1})$ (if $a(i)+1 = b(i)$, then let the x-coordinate of the right endpoint be $x(p_{a(i)+1})$). Let S' be the set of all segments thus created.

We solve the following 1D coverage problem: Find a minimum subset of segments of S' that together cover all points of P'. This problem can be easily solved in $O((|S'| + |P'|)\log(|S'| + |P'|))$ time [21],[1] which is $O((m+n)\log(m+n))$ since $|P'| = n$ and $|S'| \leq m$.
Suppose S'_1 is any optimal solution to the above 1D coverage problem. We create a subset S_1 of S^* as follows. For each segment of S'_1, suppose it is created from a disk $s_i \in S^*$; then we add s_i to S_1. We prove in the full version of the paper that S_1 is an optimal solution to the coverage problem for S^* and P.

We summarize the result in the following theorem.

▶ **Theorem 3.** *Given a set P of n points and a set S of m disks in the plane such that the disk centers are separated from points of P by a line, and the single-intersection condition is satisfied, the disk coverage problem for P and S is solvable in $O(m\sqrt{n} + (n+m)\log(n+m))$ time.*

The unit-disk case. In Section 4, we will reduce the time to $m^{2/3}n^{2/3}2^{O(\log^*(m+n))} + O((n+m)\log(n+m))$ for the unit-disk case. The algorithm is exactly the same as above, except that we compute $a(i)$'s and $b(i)$'s in a more efficient way by utilizing the property that all disks have the same radius.

3.1 Algorithm implementation

In this section, we show that the first main step of the algorithm can be implemented in $O(m\sqrt{n} + (n+m)\log(n+m))$ time. The goal is to compute $a(i)$ and $b(i)$ for all disks $s_i \in S$. We only discuss how to compute $a(i)$ since computing $b(i)$ can be done analogously. To this end, we start with the following definition.

▶ **Definition 4.** *For each point $p \in P$, define $\gamma(p)$ as the smallest index k such that the disk s_k covers p.*

One reason we introduce $\gamma(p)$ is due to the following observation.

▶ **Observation 5.** *For any disk $s_i \in S$ and any point $p \in P$ that is outside s_i, there is a disk in $S_l(s_i)$ covering p if and only if $\gamma(p) < i$.*

Our algorithm for computing $a(i)$ relies on $\gamma(p)$ for all $p \in P$. Therefore, we first present an algorithm in the following lemma to compute $\gamma(p)$.

▶ **Lemma 6.** *There is an algorithm that can compute $\gamma(p)$ for all $p \in P$ in $O(m\sqrt{n} + (m+n)\log(m+n))$ time.*

Proof. Let H be the set of the upper arcs of all disks. As discussed in Section 2, we compute a hierarchical $(1/r)$-cutting $\Xi_0, \ldots, \Xi_k$ for H in $O(mr)$ time [10,22], for a parameter $r \in [1, m]$ to be determined later. We follow the notation about cutting as in Section 2, e.g., Ξ, H_σ, S_σ, etc. Recall that Ξ denotes the set of all cells of all cuttings σ_i, $i = 0, 1, \ldots, k$. As discussed in Section 2, the cutting algorithm [10, 22] also computes the conflict lists H_σ (and thus S_σ) for all cells $\sigma \in \Xi$. Also, $\sum_{\sigma\in\Xi} |S_\sigma| = O(mr)$.

[1] The algorithm in [21], which uses dynamic programming, is for the weighted case where each segment has a weight. Our problem is simpler since it is an unweighted case. We can use a simple greedy algorithm to solve it.

For each i with $1 \leq i \leq k$, for each cell $\sigma \in \Xi_i$, let $S(\sigma)$ be the set of disks that contain σ but do not contain σ', where σ' is the parent cell of σ (which is in Ξ_{i-1}). Note that Ξ_0 consists of a single cell σ^* that is the entire plane and thus we simply let $S(\sigma^*) = \emptyset$ as no disk contains the entire plane.

We can compute $S(\sigma)$ of all cells $\sigma \in \Xi$ in $O(mr)$ time as follows. For each i with $1 \leq i \leq k$, for each cell $\sigma' \in \Xi_{i-1}$, recall that $S_{\sigma'}$ is available from the cutting algorithm. For each disk s of $S_{\sigma'}$, for each child cell σ of σ', we check whether s contains σ; if yes, we add s to $S(\sigma)$. As such, since the total size of S_σ of all cells σ of Ξ is $O(mr)$ and each cell has $O(1)$ children, the total time for computing $S(\sigma)$ for all cells $\sigma \in \Xi$ is $O(mr)$.

For each cell σ, by slightly abusing the notation, we define $\gamma(\sigma)$ as the smallest index of the disks in $S(\sigma)$. After $S(\sigma)$'s are computed, the indices $\gamma(\sigma)$ for all cells $\sigma \in \Xi$ can be computed in additional $O(mr)$ time.

Next, we run the following *point location step* for each point $p \in P$ to compute $\gamma(p)$. Initially, we set $\gamma(p) = m + 1$. Starting from the only cell of Ξ_0, we locate the cell σ_i that contains p in each cutting Ξ_i. This takes $O(\log r)$ time as each cell contains $O(1)$ children and $k = O(\log r)$. For each such cell σ_i, we update $\gamma(p) = \min\{\gamma(p), \gamma(\sigma_i)\}$. As such, the point location step on p takes $O(\log r)$ time. The total time for all points of P is $O(n \log r)$.

In addition, we do the following processing for the cell σ_k of the last cutting Ξ_k that contains each $p \in P$. For each disk $s_j \in S_{\sigma_k}$, we check whether s_j contains p. If yes, we update $\gamma(p) = \min\{\gamma(p), j\}$. After that, $\gamma(p)$ is correctly computed. As $|S_{\sigma_k}| \leq m/r$, this additional step for each point p takes $O(m/r)$ time. Therefore, the total time of this step for all points of P is $O(nm/r)$.

In summary, computing $\gamma(p)$ for all $p \in P$ takes $O(mr + n \log r + nm/r)$ time. Setting $r = \min\{\sqrt{n}, m\}$ leads to the lemma. ◀

The following lemma finally computes $a(i)$.

▶ **Lemma 7.** *Computing $a(i)$ for all disks $s_i \in S$ can be done in $O(m\sqrt{n}+(m+n)\log(m+n))$ time.*

Proof. We first compute $\gamma(p)$ for all $p \in P$ by Lemma 6.

Let H be the set of the upper arcs of all disks. As discussed in Section 2, we compute a hierarchical $(1/r)$-cutting $\Xi_0, \ldots, \Xi_k$ for H in $O(mr)$ time [10, 22], for a parameter $r \in [1, m]$ to be determined later. We follow the notation about cutting as in Section 2, e.g., Ξ, H_σ, S_σ, etc. Recall that Ξ denotes the set of all cells of all cuttings σ_i, $i = 0, 1, \ldots, k$.

For each cell $\sigma \in \Xi$, let $P(\sigma)$ denote the set of points of P inside σ, i.e., $P(\sigma) = P \cap \sigma$. We can compute $P(\sigma)$ for all cells $\sigma \in \Xi$ in $O(n \log r)$ time by the point location step as discussed in Lemma 6. Note that the total size of $P(\sigma)$ for all cells $\sigma \in \Xi$ is also $O(n \log r)$. In addition, if we invoke the point location step for points of P following their index order, then points in each $P(\sigma)$ can be sorted in their index order and the time is still $O(n \log r)$.

We need to perform a *pruning procedure* for $P(\sigma)$ of each cell $\sigma \in \Xi$. Before we describe it, we first explain the motivation. Our algorithm for computing $a(i)$ needs to solve the following subproblem. Given a disk s_i and a cell $\sigma \in \Xi$ such that σ does not intersect s_i, the problem is to compute $a_\sigma(i)$, which is defined as the largest index k of a point p_k of $P(\sigma)$ with $\gamma(p_k) < i$ (if no such k exists, then $a_\sigma(i) = 0$). In light of Observation 5, $a_\sigma(i)$ is the largest index k of a point p_k of $P(\sigma)$ such that $S_l(s_i)$ has a disk covering p_k. To solve the subproblem, consider two points p_k and p_j in $P(\sigma)$ with $k < j$. A *key observation* is that if $\gamma(p_k) \geq \gamma(p_j)$, then $a_\sigma(i) \neq k$ holds for any such disk s_i with $s_i \cap \sigma = \emptyset$, and thus p_k can simply be ignored. Indeed, assume to the contrary that $a_\sigma(i) = k$. Then, we have

$\gamma(p_k) < i$. Hence, $\gamma(p_j) < i$. By definition, we can obtain $a_\sigma(i) \geq j > k$, which contradicts with $a_\sigma(i) = k$. In light of the key observation, to facilitate computing $a_\sigma(i)$ for all such disks s_i, we first perform the following pruning procedure.

The algorithm maintains a stack A of points of $P(\sigma)$. Initially, $A = \emptyset$. We process the points of $P(\sigma)$ in their index order (recall that they are already sorted). Suppose we are processing a point $p \in P(\sigma)$. Let p' be the point at the top of the stack. If $A = \emptyset$ or if $\gamma(p') < \gamma(p)$, then we push p onto A. Otherwise, we pop p' out of A (we say that p' is pruned) and repeat the above. After all points of $P(\sigma)$ are processed, let $P'(\sigma)$ denote the set of points in the stack. Due to the pruning, points of $P'(\sigma)$ are sorted by both their indices and their $\gamma(\cdot)$ values. Clearly, the pruning procedure runs in $O(|P(\sigma)|)$ time.

We use $P'(\sigma)$ in the following way. Recall that we wish to compute $a_\sigma(i)$. Let k be the largest index of $p_k \in P'(\sigma)$ such that $\gamma(p_k) < i$. Then, the above key observation and our pruning procedure guarantee that $a_\sigma(i) = k$. Hence, we could compute $a_\sigma(i)$ by a binary search on $P'(\sigma)$ using i, the index of the disk. However, doing binary search for each disk would make the total runtime of the algorithm have one more logarithmic factor. To improve it, we use the following strategy. For each cell σ, suppose $S'(\sigma)$ is a set of disks s_i (with $s_i \cap \sigma = \emptyset$) that need to compute $a_\sigma(i)$ with respect to σ (the exact definition of $S'(\sigma)$ will be given later). Then, we search $P'(\sigma)$ with disks of $S'(\sigma)$ altogether, by using a procedure similar to that for merging two sorted lists in merge-sort. In this way, the total time is linear in $|P'(\sigma)| + |S'(\sigma)|$ (in contrast, the time would be $O(|S'(\sigma)| \cdot \log |P'(\sigma)|)$ if we do binary search for each disk of $S'(\sigma)$).

We are now ready to describe our overall algorithm for computing $a(i)$. The above has computed $P(\sigma)$ for all cells $\sigma \in \Xi$, whose total size is $O(n \log r)$. We run the pruning procedure on $P(\sigma)$ for every cell $\sigma \in \Xi$ to compute $P'(\sigma)$; this takes $O(n \log r)$ time in total as $\sum_{\sigma \in \Xi} |P(\sigma)| = O(n \log r)$.

For each cell $\sigma \in \Xi$, we define $S'(\sigma)$ as the subset of disks that do not intersect σ but whose upper arcs intersect the parent cell of σ. We can compute $S'(\sigma)$ for all cells $\sigma \in \Xi$ in $O(mr)$ time as follows. Initially we set $S'(\sigma) = \emptyset$. Then, for each $0 \leq i \leq k-1$, for each cell $\sigma' \in \Xi_i$, for each disk $s \in S_{\sigma'}$, for each child σ of σ', if s does not intersect σ, then we add s to $S'(\sigma)$. As each cell σ' has $O(1)$ children and $\sum_{\sigma \in \Xi} |S_\sigma| = O(mr)$, it takes $O(mr)$ time to compute $S'(\sigma)$ for all cells $\sigma \in \Xi$. This also implies $\sum_{\sigma \in \Xi} |S'(\sigma)| = O(mr)$.

Now that we have $P'(\sigma)$ and $S'(\sigma)$ available for all cells $\sigma \in \Xi$, we compute $a(i)$ for all disks s_i as follows. Initially, we set $a(i) = 0$. Then, for each cell σ, we perform a search with $P'(\sigma)$ and $S'(\sigma)$ to compute $a_\sigma(i)$ for all disks $s_i \in S'(\sigma)$ using the procedure discussed above, which takes $O(|P'(\sigma)| + |S'(\sigma)|)$ time. Then, for each disk $s_i \in S'(\sigma)$, we update $a(i) = \max\{a(i), a_\sigma(i)\}$. Since $\sum_{\sigma \in \Xi} |P'(\sigma)| = O(n \log r)$ and $\sum_{\sigma \in \Xi} |S(\sigma)| = O(mr)$, processing all cells σ of Ξ as above takes $O(mr + n \log r)$ time in total.

Finally, for each cell σ of the last cutting Ξ_k, we perform the following additional processing: For each disk $s_i \in S_\sigma$, for each point $p_j \in P(\sigma)$, if p_j is outside s_i and $\gamma(p_j) < i$, then we update $a(i) = \max\{a(i), j\}$. After that, the values $a(i)$ for all disks $s_i \in S$ are correctly computed. Since $|S_\sigma| \leq m/r$ for each cell $\sigma \in \Xi_k$, we spend $O(m/r)$ time on each point $p \in P(\sigma)$. As $\sum_{\sigma \in \Xi_k} |P(\sigma)| = n$, the total time of the additional processing as above for all cells $\sigma \in \Xi_k$ is $O(nm/r)$.

In summary, we can compute $a(i)$ for all disks $s_i \in S$ in $O(mr + n \log r + nm/r)$ time in total. Setting $r = \min\{\sqrt{n}, m\}$ leads to the lemma. ◀

4 The unit-disk case

In this section, we consider the unit-disk case where all disks of S have the same radius. As remarked right after Theorem 3, our algorithm is the same as described in Section 3, except that we are able to compute $a(i)$'s and $b(i)$'s more efficiently in $m^{2/3}n^{2/3}2^{O(\log^*(m+n))} + O((n+m)\log(n+m))$ time, by exploring the property that all disks have the same radius. In the following, we only discuss how to compute $a(i)$'s because the case for $b(i)$'s is similar.

For each point $p_i \in P$, define $\tilde{p}_i$ as the unit disk centered at p_i, and we call $\tilde{p}_i$ the *dual disk* of p_i. For each disk s_i, let $\tilde{s}_i$ denote the center of s_i, and we call $\tilde{s}_i$ the *dual point* of s_i. Let $\tilde{P}$ denote the set of all dual disks and $\tilde{S}$ the set of all dual points. Since all disks of S have the same radius, we have the following easy observation.

▶ **Observation 8.** *A disk $s_i \in S$ contains a point $p_j \in P$ if and only if the dual point $\tilde{s}_i$ is contained in the dual disk $\tilde{p}_j$.*

Our new algorithm for computing $a(i)$'s for the unit-disk case relies on exploring the "duality" in Observation 8. Recall in Section 3.1 that the algorithm for computing $a(i)$'s involves two steps: (1) compute $\gamma(p)$'s for all points $p \in P$ (i.e., Lemma 6); (2) compute $a(i)$'s for all $s_i \in P$ (i.e., Lemma 7). We have new algorithms for both steps in the following two subsections, respectively.

4.1 Computing $\gamma(p)$'s

We first introduce the following definition $\tilde{\gamma}(\cdot)$, which is "dual" to $\gamma(\cdot)$.

▶ **Definition 9.** *For each dual disk $\tilde{p} \in \tilde{P}$, define $\tilde{\gamma}(\tilde{p})$ as the smallest index k such that $\tilde{p}$ contains the dual point $\tilde{s}_k$.*

The following observation follows immediately from Observation 8.

▶ **Observation 10.** *For each point $p_i \in P$, $\gamma(p_i) = \tilde{\gamma}(\tilde{p}_i)$.*

Observation 10 implies that computing $\gamma(p_i)$ for all points $p_i \in S$ is equivalent to computing $\tilde{\gamma}(\tilde{p}_i)$ for all dual disks $\tilde{p}_i \in \tilde{P}$. To compute them, we will present two recursive algorithms and then combine them to obtain our final algorithm. The first algorithm computes $\gamma(p_i)$'s using P and S while the second one computes $\tilde{\gamma}(\tilde{p}_i)$ using $\tilde{P}$ and $\tilde{S}$. The combined algorithm will run the two algorithms alternatively using recursion.

The first algorithm. This algorithm follows the same framework as that for Lemma 6, but when processing the cells σ in the last cutting Ξ_k, instead of brute-force, we form subproblems and solve them recursively. We follow the notation from Lemma 6.

Let H be the set of the upper arcs of all disks of S. We compute a hierarchical $(1/r)$-cutting $\Xi_0, \ldots, \Xi_k$ for H in $O(mr)$ time [10, 22], for a parameter $r \in [1, m]$ to be determined later. Let Ξ denote the set of all cells of all cuttings σ_i, $i = 0, 1, \ldots, k$. As in Lemma 6, we compute $S(\sigma)$ and $\gamma(\sigma)$ for all cells $\sigma \in \Xi$, which takes $O(mr)$ time.

Next, we run the point location step for each point $p \in P$ as in Lemma 6. Initially, we set $\gamma(p) = 0$. Starting from Ξ_0, we locate the cell σ_i that contains p in each cutting Ξ_i. For each σ_i, we update $\gamma(p) = \min\{\gamma(p), \gamma(\sigma_i)\}$. This point location step can also compute $P(\sigma)$ for all cells $\sigma \in \Xi$, where $P(\sigma)$ denotes the set of points of P in σ. The total time for the point locations for all points $p \in P$ is $O(n \log r)$.

Finally, we do the following additional processing for the last cutting Ξ_k. For each cell $\sigma \in \Xi_k$, if $|P(\sigma)| > n/r^2$, we partition $P(\sigma)$ into subsets of sizes between $n/(2r^2)$ and n/r^2, called *standard subsets* (if $|P(\sigma)| \leq n/r^2$, then $P(\sigma)$ itself is a standard subset). As Ξ_k has $O(r^2)$ cells and $\sum_{\sigma\in\Xi_k} |P(\sigma)| = n$, the total number of standard subsets for all cells $\sigma \in \Xi_k$ is $O(r^2)$. Recall that S_σ is the subset of disks whose upper arcs intersect σ. For each standard subset $P_1(\sigma)$ of $P(\sigma)$, we form a subproblem on $(P_1(\sigma), S_\sigma)$: Compute $\gamma_\sigma(p)$ for all points $p \in P_1(\sigma)$ with respect to disks of S_σ, where $\gamma_\sigma(p)$ is defined to be the smallest index of the disks of S_σ covering p. After the subproblem is solved, we update $\gamma(p) = \min\{\gamma(p), \gamma_\sigma(p)\}$ for each point $p \in P_1(\sigma)$. This will compute $\gamma(p)$ correctly. Note that there are $O(r^2)$ subproblems in total and in each subproblem $|P_1(\sigma)| \leq n/r^2$ and $|S_\sigma| \leq m/r$.

If we use $T(n, m)$ to denote the runtime of the entire algorithm on the original problem (P, S), then we obtain the following recurrence relation:

$$T(n, m) = O(mr + n \log r) + O(r^2) \cdot T(n/r^2, m/r). \tag{1}$$

The second algorithm. The second algorithm computes $\tilde{\gamma}(\tilde{p})$ for all dual disks $\tilde{p} \in \tilde{P}$.

Recall that all dual disks have their centers above ℓ. Therefore, each dual disk has a "lower arc" below ℓ. Let $\tilde{H}$ denote the set of the lower arcs of all dual disks. We compute a hierarchical $(1/r)$-cutting $\Xi_0, \ldots, \Xi_k$ for $\tilde{H}$ in $O(nr)$ time [10, 22], for a parameter $r \in [1, n]$ to be determined later. We use Ξ to denote the set of all cells of all cuttings Ξ_i, $i = 0, 1, \ldots, k$. For each cell $\sigma \in \Xi$, let $\tilde{P}_\sigma$ denote the set of dual disks whose lower arcs intersect σ.

For each cell $\sigma \in \Xi$, let $\tilde{S}(\sigma)$ denote the subset of dual points of $\tilde{S}$ inside σ. For each cell $\sigma \in \Xi$, by slightly abusing the notation, let $\tilde{\gamma}(\sigma)$ denote the minimum index of all points of $\tilde{S}(\sigma)$. We can compute $\tilde{S}(\sigma)$ as well as $\tilde{\gamma}(\sigma)$ for all cells $\sigma \in \Xi$ in $O(m \log r)$ time by point locations as in the first algorithm.

We now compute $\tilde{\gamma}(\tilde{p})$'s. Initially, we set each $\tilde{\gamma}(\tilde{p}) = m + 1$. For each $1 \leq i \leq k$, for each cell $\sigma' \in \Xi_{i-1}$, for each dual disk $\tilde{p} \in \tilde{P}_{\sigma'}$, for each child $\sigma \in \Xi_i$ of σ', if $\tilde{p}$ contains σ, then we update $\tilde{\gamma}(\tilde{p}) = \min\{\tilde{\gamma}(\tilde{p}), \tilde{\gamma}(\sigma)\}$. Since $\sum_{\sigma\in\Xi} |\tilde{P}_\sigma| = O(nr)$ and each cell has $O(1)$ children, the total time of this procedure is $O(nr)$.

Finally, we do the following additional processing for the last cutting Ξ_k. For each cell $\sigma \in \Xi_k$, as in the first algorithm, if $|\tilde{S}(\sigma)| > m/r^2$, we partition $\tilde{S}(\sigma)$ into *standard subsets* of sizes between $m/(2r^2)$ and m/r^2. The total number of standard subsets is $O(r^2)$. For each standard subset $\tilde{S}_1(\sigma)$ of $\tilde{S}(\sigma)$, we form a subproblem on $(\tilde{P}_\sigma, \tilde{S}_1(\sigma))$: Compute $\tilde{\gamma}_\sigma(\tilde{p})$ for all dual disks $\tilde{p} \in \tilde{P}(\sigma)$ with respect to the dual points of $\tilde{S}_1(\sigma)$, where $\tilde{\gamma}_\sigma(\tilde{p})$ is the smallest index of the dual points of $\tilde{S}_1(\sigma)$ contained in $\tilde{p}$. After the subproblem is solved, we update $\tilde{\gamma}(\tilde{p}) = \min\{\tilde{\gamma}(\tilde{p}), \tilde{\gamma}_\sigma(\tilde{p})\}$ for each $\tilde{p} \in \tilde{P}(\sigma)$. This will compute $\tilde{\gamma}(\tilde{p})$ correctly. Note that there are $O(r^2)$ subproblems in total and in each subproblem $|\tilde{S}_1(\sigma)| \leq m/r^2$ and $|\tilde{P}_\sigma| \leq n/r$.

Recall that $T(n, m)$ refers to our problem for computing $\gamma(p)$'s on (P, S), which is equivalent to computing $\tilde{\gamma}(\tilde{p})$'s on $(\tilde{P}, \tilde{S})$ by Observation 8. Hence, we can also obtain the following recurrence relation using the second algorithm:

$$T(n, m) = O(nr + m \log r) + O(r^2) \cdot T(n/r, m/r^2). \tag{2}$$

Combining the two algorithms. We now combine the two algorithms to compute $\gamma(p)$'s for all $p \in P$.

We first discuss the *symmetric case* where $m = n$ (if $m \neq n$, it is the *asymmetric case*). If we apply (1) and then (2) using the same r, we can obtain the following recurrence

$$T(n, n) = O(nr \log r) + O(r^4) \cdot T(n/r^3, n/r^3).$$

Setting $r = n^{1/3}/\log n$ leads to the following

$$T(n,n) = O(n^{4/3}) + O((n/\log^3 n)^{4/3}) \cdot T(\log^3 n, \log^3 n).$$

The recurrence solves to $T(n,n) = n^{4/3}2^{O(\log^* n)}$.

We next tackle the asymmetric case, by using the above symmetric case result. Depending on whether $m \geq n$, there are two cases.

1. If $m \geq n$, depending on whether $m < n^2$, there are two subcases.
   a. If $m < n^2$, then set $r = m/n$ so that $n/r = m/r^2$. Applying (2) with $r = m/n$ and solving each subproblem $T(n/r, m/r^2)$ using the symmetric case result give us $T(n,m) = m^{2/3}n^{2/3}2^{O(\log^* m)}$.
   b. If $m \geq n^2$, then applying (2) with $r = n$ gives us $T(n,m) = O(n^2 + m\log n) + O(n^2) \cdot T(1, m/n^2)$. Clearly, we have $T(1, m/n^2) = O(m/n^2)$. Hence, we obtain $T(m,n) = O(m\log n)$ since $m \geq n^2$.

   Hence in the case where $m \geq n$ we have $T(n,m) = O(m\log n) + m^{2/3}n^{2/3}2^{O(\log^* m)}$.
2. If $m < n$, the analysis is similar (using (1) instead) and we can obtain $T(n,m) = O(n\log m) + m^{2/3}n^{2/3}2^{O(\log^* n)}$.

In summary, computing $\gamma(p)$'s for all points $p \in P$ can be done in $O((n+m)\log(m+n)) + m^{2/3}n^{2/3}2^{O(\log^*(n+m))}$ time.

4.2 Computing $a(i)$'s

With $\gamma(p)$'s computed above, we describe our algorithm for computing $a(i)$'s for all disks $s_i \in S$. As in Section 4.1, we first introduce the following definition, which is "dual" to $a(i)$.

▶ **Definition 11.** *For each dual point $\tilde{s}_i \in \tilde{S}$, define $\tilde{a}(i)$ as the largest index k of the dual disk $\tilde{p}_k \in \tilde{P}$ such that $\tilde{p}_k$ contains a dual point $\tilde{s}_j$ with $j < i$ but does not contain $\tilde{s}_i$.*

Based on Observation 8, we have the following lemma.

▶ **Lemma 12.** *For each $1 \leq i \leq m$, $a(i) = \tilde{a}(i)$.*

Proof. Consider the point $p_k \in P$ with $k = a(i)$. By definition, p_k is outside s_i and S has a disk s_j that covers p_k with $j < i$. Then, by Observation 8, the dual disk $\tilde{p}_k$ contains the dual point $\tilde{s}_j$ but does not contain the dual point $\tilde{s}_i$. By definition, it must hold that $\tilde{a}(i) \geq k = a(i)$.

Analogously, we can prove that $a(i) \geq \tilde{a}(i)$. ◀

Lemma 12 implies that computing $a(i)$ for all disks $s_i \in S$ is equivalent to computing $\tilde{a}(i)$ for all dual points $\tilde{s}_i \in \tilde{S}$. To compute them, as in Section 4.1, we will also present two recursive algorithms and then combine them. The first algorithm computes $a(i)$'s using P and S while the second one computes $\tilde{a}(i)$'s using $\tilde{P}$ and $\tilde{S}$. The combined algorithm will run the two algorithms alternatively using recursion. In what follows, we assume that $\gamma(p)$ for all points $p \in P$ and $\tilde{\gamma}(\tilde{p})$ for all $\tilde{p} \in \tilde{P}$ have been computed.

The first algorithm. The first algorithm follows the framework of Lemma 7 but uses recursion when we process the last cutting Ξ_k. Here we only discuss how to perform additional preprocessing for the cells of the last cutting Ξ_k; the rest of the algorithm is the same as before, which takes $O(mr + n\log r)$ time in total. We follow the notation in the proof of Lemma 7.

For each cell $\sigma \in \Xi_k$, if $|P(\sigma)| > n/r^2$, we partition $P(\sigma)$ into *standard subsets* of sizes between $n/(2r^2)$ and n/r^2. Recall that S_σ is the subset of disks of S whose upper arcs intersect σ. For each standard subset $P_1(\sigma)$ of $P(\sigma)$, we form a subproblem on $(P_1(\sigma), S_\sigma)$: Compute $a_\sigma(i)$ for all disks $s_i \in S(\sigma)$ with respect to points of $P_1(\sigma)$, where $a_\sigma(i)$ is the largest index k of a point $p_k \in P_1(\sigma)$ that is outside s_i but is covered by a disk s_j with $j < i$. After the subproblem is solved, we update $a(i) = \max\{a(i), a_\sigma(i)\}$ for each disk $s_i \in S(\sigma)$. This will compute $a(i)$ correctly. Note that there are $O(r^2)$ subproblems in total and in each subproblem $|P_1(\sigma)| \leq n/r^2$ and $|S_\sigma| \leq m/r$.

If we use $T(n, m)$ to denote the runtime of the entire algorithm on the original problem (P, S), then we obtain the following recurrence relation:

$$T(n, m) = O(mr + n \log r) + O(r^2) \cdot T(n/r^2, m/r). \tag{3}$$

The second algorithm. The second algorithm computes $\tilde{a}(i)$ for all dual points $\tilde{s}_i \in S$.

Recall that all dual disks have their centers above ℓ. Therefore, each dual disk has a "lower arc" below ℓ. Let $\tilde{H}$ denote the set of lower arcs of all dual disks. We compute a hierarchical $(1/r)$-cutting $\Xi_0, \ldots, \Xi_k$ for $\tilde{H}$ in $O(nr)$ time [10, 22], for a parameter $r \in [1, n]$ to be determined later. We use Ξ to denote the set of cells of all cuttings Ξ_i, $i = 0, 1, \ldots, k$. For each cell $\sigma \in \Xi$, let $\tilde{P}_\sigma$ denote the set of dual disks whose lower arcs intersect σ.

For each cell $\sigma \in \Xi$, let $\tilde{S}(\sigma)$ be the set of dual points of $\tilde{S}$ inside σ. We can compute $\tilde{S}(\sigma)$ for all cells of Ξ in $O(m \log r)$ time using point locations as discussed before. Also, $\sum_{\sigma \in \Xi} |\tilde{S}(\sigma)| = O(m \log r)$. In addition, points in each $\tilde{S}(\sigma)$ can be sorted in their index order if we invoke the point location step on dual points of $\tilde{S}$ in their index order; the total time is still $O(m \log r)$.

For each cell $\sigma \in \Xi$, we define $\tilde{P}(\sigma)$ in the same way as $S'(\sigma)$ in the proof of Lemma 7. Specifically, $\tilde{P}(\sigma)$ is the subset of dual disks of $\tilde{P}$ that do not intersect σ but whose lower arcs intersect the parent cell of σ. As in Lemma 7, $\tilde{P}(\sigma)$ for all $\sigma \in \Xi$ can be computed in $O(nr)$ time and $\sum_{\sigma \in \Xi} |\tilde{P}(\sigma)| = O(nr)$.

Now consider the following problem on $\tilde{S}(\sigma)$ and $\tilde{P}(\sigma)$. For each dual point $\tilde{s}_i \in \tilde{S}(\sigma)$, we want to compute $\tilde{a}_\sigma(i)$, which is the largest index k of a dual disk $\tilde{p}_k \in \tilde{P}(\sigma)$ that contains a dual point $\tilde{s}_j$ with $j < i$. After solving the problem, we update $\tilde{a}(i) = \max\{\tilde{a}(i), \tilde{a}_\sigma(i)\}$. To solve the problem, first notice that $\tilde{p}_k$ contains a dual point $\tilde{s}_j$ with $j < i$ if and only if $\tilde{\gamma}(\tilde{p}_k) < i$. Then, consider two dual disks $\tilde{p}_k$ and $\tilde{p}_j$ in $\tilde{P}(\sigma)$ with $k < j$. A *key observation* is that if $\tilde{\gamma}(k) \geq \tilde{\gamma}(j)$, then $\tilde{a}_\sigma(i) \geq j$ holds for any dual point $\tilde{s}_i \in \tilde{S}(\sigma)$ (and thus $\tilde{p}_k$ can be ignored; this echoes the key observation in Lemma 7).

Using the key observation, as in the proof of Lemma 7, we run a pruning procedure on $\tilde{P}(\sigma)$ to obtain a subset $\tilde{P}'(\sigma)$ of dual disks that are sorted both by their indices and their $\tilde{\gamma}(\cdot)$ values. The pruning procedure takes $O(|\tilde{P}(\sigma)|)$ time if dual disks of $\tilde{P}(\sigma)$ are already sorted by their indices. We can produce the sorted lists of $\tilde{P}(\sigma)$ for all cells $\sigma \in \Xi$ in $O(nr)$ time as follows. First, for each dual disk $\tilde{p}_i$, we create a list L_i that contains all cells $\sigma \in \Xi$ such that $\tilde{p}_i$ is in $\tilde{P}_\sigma$. This can be done in $O(nr)$ time by traversing the conflict lists of all cells. Second, we process the lists $L_1, L_2, \ldots, L_n$ in this order. For each list L_i, for each cell $\sigma \in L_i$, we add $\tilde{p}_i$ to the rear of a list $L(\sigma)$ for σ (initially, $L(\sigma) = \emptyset$). Once all lists $L_1, L_2, \ldots, L_n$ are processed as above, $L(\sigma)$ contains the sorted list of the dual disks of $\tilde{P}_\sigma$ by their indices. The total time of this sorting algorithm is linear in $\sum_{\sigma \in \Xi} |\tilde{P}_\sigma|$, which is $O(nr)$.

After the pruning procedure, we proceed with $\tilde{P}(\sigma)$ as follows. Suppose k is the largest index of any dual disk $\tilde{p}_k \in \tilde{P}(\sigma)$ such that $\tilde{\gamma}(\tilde{p}_k) < i$; then we have $\tilde{a}_\sigma(\tilde{s}_i) = k$. As such, we can scan the two lists $\tilde{P}(\sigma)$ and $\tilde{S}(\sigma)$ simultaneously (recall that dual points of $\tilde{S}(\sigma)$ are

also sorted by their indices), which can compute $\tilde{a}_\sigma(\tilde{s}_i)$'s for all dual points $\tilde{s}_i \in \tilde{S}(\sigma)$ in $O(|\tilde{P}(\sigma)| + |\tilde{S}(\sigma)|)$ time. As $\sum_{\sigma\in\Xi} |\tilde{P}(\sigma)| = O(nr)$ and $\sum_{\sigma\in\Xi} |\tilde{S}(\sigma)| = O(m \log r)$, the total time for doing this for all cells $\sigma \in \Xi$ is $O(m \log r + nr)$.

Finally, we do the following additional processing for the last cutting Ξ_k. For each cell $\sigma \in \Xi_k$, if $|\tilde{S}(\sigma)| > m/r^2$, we partition $\tilde{S}(\sigma)$ into *standard subsets* of sizes between $m/(2r^2)$ and m/r^2. Recall that $\tilde{P}_\sigma$ is the subset of dual disks whose lower arcs intersect σ. For each standard subset $\tilde{S}_1(\sigma)$ of $\tilde{S}(\sigma)$, we form a subproblem on $(\tilde{P}_\sigma, \tilde{S}_1(\sigma))$: Compute $\tilde{a}_\sigma(i)$ for all dual points $\tilde{s}_i \in \tilde{S}_1(\sigma)$ with respect to dual disks of $\tilde{P}_\sigma$, where $\tilde{a}_\sigma(i)$ is the largest index k of a dual disk $\tilde{p}_k \in \tilde{P}_\sigma$ that contains a dual point $\tilde{s}_j$ with $j < i$ but does not contain $\tilde{s}_i$. After the subproblem is solved, we update $\tilde{a}(i) = \max\{\tilde{a}(i), \tilde{a}_\sigma(i)\}$ for each dual point $\tilde{s}_i \in \tilde{S}_1(\sigma)$. This will compute $\tilde{a}(i)$ correctly. Note that there are $O(r^2)$ subproblems in total and in each subproblem $|\tilde{P}_\sigma| \le n/r$ and $|\tilde{S}_1(\sigma)| \le m/r^2$.

Recall that $T(n, m)$ refers to our problem for computing $a(i)$'s on (P, S), which is equivalent to computing $\tilde{a}(i)$'s on $(\tilde{P}, \tilde{S})$ by Lemma 12. Hence, we can obtain the following recurrence relation using the second algorithm:

$$T(n, m) = O(nr + m \log r) + O(r^2) \cdot T(n/r, m/r^2). \tag{4}$$

Combining the two algorithms. Following exactly the same approach and the same analysis as in Section 4.1 and using (3) and (4), we can obtain a combined algorithm that can compute $a(i)$ for all disks $s_i \in S$ in $O((n + m) \log(n + m)) + m^{2/3}n^{2/3}2^{O(\log^*(n+m))}$ time.

We summarize our result in the following theorem.

▶ **Theorem 13.** *Given a set P of n points and a set S of m unit disks in the plane such that the disk centers are separated from points of P by a line, the disk coverage problem for P and S is solvable in $O((n + m) \log(n + m)) + m^{2/3}n^{2/3}2^{O(\log^*(n+m))}$ time.*

References

1. Pankaj K. Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. *Discrete and Computational Geometry*, 63:460–482, 2020.
2. Helmut Alt, Esther M. Arkin, Hervé Brönnimann, Jeff Erickson, Sándor P. Fekete, Christian Knauer, Jonathan Lenchner, Joseph S. B. Mitchell, and Kim Whittlesey. Minimum-cost coverage of point sets by disks. In *Proceedings of the 22nd Annual Symposium on Computational Geometry (SoCG)*, pages 449–458, 2006.
3. Christoph Ambühl, Thomas Erlebach, Matúš Mihalák, and Marc Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In *Proceedings of the 9th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), and the 10th International Conference on Randomization and Computation (RANDOM)*, pages 3–14, 2006.
4. Vittorio Bilò, Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos. Geometric clustering to minimize the sum of cluster sizes. In *Proceedings of the 13th European Symposium on Algorithms (ESA)*, pages 460–471, 2005.
5. Ahmad Biniaz, Prosenjit Bose, Paz Carmi, Anil Maheshwari, J.Ian Munro, and Michiel Smid. Faster algorithms for some optimization problems on collinear points. In *Proceedings of the 34th International Symposium on Computational Geometry (SoCG)*, pages 8:1–8:14, 2018.
6. Norbert Bus, Nabil H. Mustafa, and Saurabh Ray. Practical and efficient algorithms for the geometric hitting set problem. *Discrete Applied Mathematics*, 240:25–32, 2018.
7. Paz Carmi, Matthew J. Katz, and Nissan Lev-Tov. Covering points by unit disks of fixed location. In *Proceedings of the International Symposium on Algorithms and Computation (ISAAC)*, pages 644–655, 2007.

8 Timothy M. Chan and Elyot Grant. Exact algorithms and APX-hardness results for geometric packing and covering problems. *Computational Geometry: Theory and Applications*, 47:112–124, 2014.

9 Timothy M. Chan and Qizheng He. Faster approximation algorithms for geometric set cover. In *Proceedings of 36th International Symposium on Computational Geometry (SoCG)*, pages 27:1–27:14, 2020.

10 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9(2):145–158, 1993.

11 Francisco Claude, Gautam K. Das, Reza dorrigiv, Stephane Durocher, Robert Fraser, Alejandro López-Ortiz, Bradford G. Nickerson, and Alejandro Salinger. An improved line-separable algorithm for discrete unit disk cover. *Discrete Mathematics, Algorithms and Applications*, 2:77–88, 2010.

12 Gruia Călinescu, Ion I. Măndoiu, Peng-Jun Wan, and Alexander Z. Zelikovsky. Selecting forwarding neighbors in wireless ad hoc networks. *Mobile Networks and Applications*, 9:101–111, 2004.

13 Gautam K. Das, Sandip Das, and Subhas C. Nandy. Homogeneous 2-hop broadcast in 2D. *Computational Geometry: Theory and Applications*, 43:182–190, 2010.

14 Herbert Edelsbrunner and Ernst P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9:66–104, 1990.

15 Tomás Feder and Daniel H. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 434–444, 1988.

16 Shashidhara K. Ganjugunte. *Geometric hitting sets and their variants*. PhD thesis, Duke University, 2011.

17 Sariel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *Journal of Computational Geometry*, 3:65–85, 2012.

18 Nissan Lev-Tov and David Peleg. Polynomial time approximation schemes for base station coverage with minimum total radii. *Computer Networks*, 47:489–501, 2005.

19 Jian Li and Yifei Jin. A PTAS for the weighted unit disk cover problem. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 898–909, 2015.

20 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44:883–895, 2010.

21 Logan Pedersen and Haitao Wang. Algorithms for the line-constrained disk coverage and related problems. *Computational Geometry: Theory and Applications*, 105-106:101883:1–18, 2022.

22 Haitao Wang. Unit-disk range searching and applications. In *Proceedings of the 18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 32:1–32:17, 2022.

Improved Smoothed Analysis of 2-Opt for the Euclidean TSP

Bodo Manthey
Faculty of Electrical Engineering, Mathematics, and Computer Science, University of Twente, Enschede, The Netherlands

Jesse van Rhijn
Faculty of Electrical Engineering, Mathematics, and Computer Science, University of Twente, Enschede, The Netherlands

Abstract

The 2-opt heuristic is a simple local search heuristic for the Travelling Salesperson Problem (TSP). Although it usually performs well in practice, its worst-case running time is poor. Attempts to reconcile this difference have used smoothed analysis, in which adversarial instances are perturbed probabilistically. We are interested in the classical model of smoothed analysis for the Euclidean TSP, in which the perturbations are Gaussian. This model was previously used by Manthey & Veenstra, who obtained smoothed complexity bounds polynomial in n, the dimension d, and the perturbation strength σ^{-1}. However, their analysis only works for $d \geq 4$. The only previous analysis for $d \leq 3$ was performed by Englert, Röglin & Vöcking, who used a different perturbation model which can be translated to Gaussian perturbations. Their model yields bounds polynomial in n and σ^{-d}, and super-exponential in d. As the fact that no direct analysis exists for Gaussian perturbations that yields polynomial bounds for all d is somewhat unsatisfactory, we perform this missing analysis. Along the way, we improve all existing smoothed complexity bounds for Euclidean 2-opt with Gaussian perturbations.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Approximation algorithms analysis; Theory of computation → Discrete optimization

Keywords and phrases Travelling salesman problem, smoothed analysis, probabilistic analysis, local search, heuristics, 2-opt

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.52

Related Version *Full Version*: `https://arxiv.org/abs/2211.16908`

Funding *Jesse van Rhijn*: Supported by NWO grant OCENW.KLEIN.176.

Acknowledgements We thank Ashkan Safari and Tjark Vredeveld for many useful discussions.

1 Introduction

The Travelling Salesperson problem is a standard combinatorial optimization problem, which has attracted considerable interest from academic, educational and industrial directions. It can be stated rather compactly: given a Hamiltonian graph $G = (V, E)$ and edge weights $w : E \to \mathbb{R}$, find a minimum weight Hamiltonian cycle (tour) on G.

Despite this apparent simplicity, the TSP is NP-hard [6]. A particularly interesting variant of the TSP is the Euclidean TSP, in which the n vertices of the graph are identified with a point cloud in $\mathbb{R}^d$, and the edge weights are the Euclidean distances between these points. Even this restricted variant is NP-hard [10].

As a consequence of this hardness, practitioners often turn to heuristics. One often-used heuristic is 2-opt [1]. This heuristic takes as its input a tour T, and finds two sets of two edges each, $\{e_1, e_2\} \subseteq T$ and $\{f_1, f_2\} \nsubseteq T$, such that exchanging $\{e_1, e_2\}$ for $\{f_1, f_2\}$ yields

again a tour T', and the total weight of T' is strictly less than the total weight of T. This procedure is repeated with the new tour, and stops once no such edges exist. The resulting tour is said to be locally optimal.

Englert, Röglin and Vöcking constructed Euclidean TSP instances on which 2-opt can take exponentially many steps to find a locally optimal tour [4]. Despite this pessimistic result, 2-opt performs remarkably well in practice, usually requiring time sub-quadratic in n and obtaining tours which are only a few percent worse than the optimum [1, chapter 8].

To explain this discrepancy, the tools of probabilistic analysis have been employed [9, 2, 5, 3, 4]. In particular, smoothed analysis, a hybrid framework between worst-case and average-case analysis, has been successfully used in the analysis of 2-opt [5, 4, 9]. In the original version of this framework, the instances one considers are initially adversarial, and then perturbed by Gaussians. The resulting smoothed time complexity is then generally a function of the instance size n and the standard deviation of the Gaussian perturbations, σ.

Englert et al. obtained smoothed time complexity bounds for 2-opt on Euclidean instances by considering a more general model, in which the points are chosen in the unit hypercube according to arbitrary probability densities. The only restrictions to these densities are that (i) they are independent, and (ii) they are all bounded from above by ϕ. Their results can be transferred to Gaussian perturbations roughly by setting $\phi = \sigma^{-d}$, which yields a smoothed complexity that is $O(\text{poly}(n, \sigma^{-d}))$.

As the exponential dependence on d is somewhat unsatisfactory, Manthey & Veenstra [9] performed a simpler smoothed analysis yielding bounds polynomial in n, $1/\sigma$, and d. The analysis they performed is however limited to $d \geq 4$. While polynomial bounds for all d can be obtained by simply taking the result of Englert et al. for $d \in \{2, 3\}$, no smoothed analysis that directly uses Gaussian perturbations exists for these cases. We set out to perform this missing analysis, improving the smoothed complexity bounds for all $d \geq 2$ along the way.

Our analysis combines ideas from both Englert et al. and Manthey & Veenstra. From the former, we borrow the idea of conditioning on the outcomes of some of the distances between points in an arbitrary 2-change. We can then analyze the 2-change by examining the angles between certain edges in the 2-change, which are themselves random variables. From the latter, we borrow the Gaussian perturbation model (originally introduced by Spielman & Teng for the Simplex Method [11]).

We also note that in addition to improving the results of Manthey & Veenstra, our approach is significantly simpler than the analysis of Englert et al. The crux of the simplification is a carefully constructed random experiment to model a single 2-change, which allows us to bypass the need for the involved convolution integrals used by Englert et al.

We will begin by introducing some definitions and earlier results, before providing basic probability theoretical results (Section 2) that we will make heavy use of throughout the paper. We then proceed by analyzing a single 2-change in a similar manner as Englert et al., simplifying some of their analysis in the process (Section 3). Next, we prove a first smoothed complexity bound by examining so-called linked pairs of 2-changes (Section 4), an idea used by both Englert et al. and Manthey & Veenstra. Finally, we improve on this bound for $d \geq 3$ (Section 5), yielding the best known bounds for all dimensions.

2 Preliminaries

2.1 Travelling Salesperson Problem

Let $\mathcal{Y} \subseteq [-1, 1]^d$ be a point set of size n. The Euclidean Travelling Salesperson Problem (TSP) asks for a tour that visits each point $y \in \mathcal{Y}$ exactly once, such that the total length of the tour is minimized. The length of a tour in this variant of the TSP is the sum of the

Euclidean distances between consecutive points in the tour. Formally, if the points in $\mathcal{Y}$ are visited in the order $T = (y_{\pi(i)})_{i=0}^{n-1}$ defined by a permutation π of $[n]$, then the length of the tour T is

$$L(T) = \sum_{i=0}^{n-1} \|y_{\pi(i)} - y_{\pi(i+1)}\|,$$

where the indices are taken modulo n, and $\|\cdot\|$ denotes the standard Euclidean norm in $\mathbb{R}^d$. Since the Euclidean TSP is undirected, the tour T' in which the vertices are visited in the reverse order has the same length as T. We consider these tours to be identical.

2.2 Smoothed Analysis

Smoothed analysis is a framework for the analysis of algorithms, which was introduced in 2004 by Spielman & Teng [11]. The method is particularly suitable to algorithms with a fragile worst case input [7]. Since its introduction, the method has been applied to a wide variety of algorithms [8, 12].

Heuristically, one imagines that an adversary chooses an input to the algorithm. The input is then perturbed in a probabilistic fashion. The hope is that any particularly pathological instances that the adversary might choose are destroyed by the random perturbation. One then computes a bound on the expected number of steps that the algorithm performs, where the expectation is taken with respect to the perturbation.

For our model of a smoothed TSP instance, we allow the adversary to choose a point set $\mathcal{Y} \subseteq [-1, 1]^d$ of size n. We then perturb each point $y_i \in \mathcal{Y}$ with an independent d-dimensional Gaussian random variable g_i, $i \in [n]$, with mean 0 and standard deviation σ. This yields a new point set, $\mathcal{X} = \{y_i + g_i \mid y_i \in \mathcal{Y}\}$. We will bound the expected number of steps taken by the 2-opt heuristic on the TSP instance defined by $\mathcal{X}$, with the expectation taken over this Gaussian perturbation. We will refer to this quantity as the smoothed complexity of 2-opt.

For the purposes of our analysis, we always assume that $\sigma \leq 1$. This is a mild restriction, as the bound for $\sigma = 1$ also applies to all larger values of σ, and small perturbations are particularly interesting in smoothed analysis.

For a general outline of the strategy, consider a 2-change where the edges $\{a, z_1\}$ and $\{b, z_2\}$ are replaced by $\{a, z_2\}$ and $\{b, z_1\}$. The change in tour length of this 2-change is

$$\Delta = \|a - z_1\| + \|b - z_2\| - \|a - z_2\| - \|b - z_1\|.$$

Since the locations of the points $\{a, b, z_1, z_2\}$ are random variables, so is Δ. We seek to bound the probability that there exists a 2-change whose improvement is exceedingly small, enabling us to use a potential argument.

Let $\Delta_{\min}$ denote the improvement of the least-improving 2-change in the instance. If $\mathbb{P}(\Delta_{\min} \leq \epsilon)$ is suitably small for small ϵ, then each iteration is likely to decrease the tour length by a large amount. As long as the initial tour has bounded length, this then provides a limit to the number of iterations that the heuristic can perform, since the tour length is bounded from below by 0.

2.3 Basic Results

We state some general results that we will need at points throughout the paper.

The next lemma provides a simple framework that we can use to prove smoothed complexity bounds for 2-opt.

Let $\Delta_{\min}$ denote the smallest improvement of any 2-change, and let $\Delta_{\min}^{\text{link}}$ denote the smallest improvement of any pair of linked 2-changes (see Section 4 for a definition of linked pairs).

▶ **Lemma 1** ([9, Lemma 2.2]). *Suppose that the longest tour has a length of at most L with probability at least $1 - 1/n!$. Let $\alpha > 1$ be a constant. If for all $\epsilon > 0$ it holds that $\mathbb{P}(\Delta_{\min} \in (0, \epsilon]) = O(P\epsilon^{\alpha})$, then the smoothed complexity of 2-opt is bounded from above by $O(P^{1/\alpha}L)$. The same holds if we replace $\Delta_{\min}$ by $\Delta_{\min}^{\text{link}}$, provided that $P^{1/\alpha}L = \Omega(n^2)$.*

2.4 Probability Theory

We provide some basic probability theoretical results. Throughout the paper, given a random variable X, we denote its probability density by f_X and its cumulative distribution function by F_X. If we furthermore condition on some event Y, we write $f_{X|Y}$ for the conditional density of X given Y.

2.4.1 Chi Distributions

Suppose we are given two points $y_1, y_2 \in \mathcal{Y}$ and perturb both points with independent Gaussian random variables g_1 and g_2, resulting in $x_i = y_i + g_i$, $i \in [2]$. Then the distance $\|x_1 - x_2\|$ between the two perturbed points is distributed according to a noncentral d-dimensional chi distribution with noncentrality parameter $s = \|y_1 - y_2\|$, which we denote χ_d^s. We call χ_d^0 a central d-dimensional χ distribution.

2.4.2 General Results

In the following, we use the notion of stochastic dominance. Let X and Y be two real-valued random variables. We say that X stochastically dominates Y if for all x, it holds that $\mathbb{P}(X \geq x) \geq \mathbb{P}(Y \geq x)$, and this inequality is strict for some x. We may equivalently say that the density of X stochastically dominates the density of Y.

To use Lemma 1, we need to limit the probability that any TSP tour in our smoothed instance is too long. This was previously done by Manthey & Veenstra; we state their result in Lemma 2.

▶ **Lemma 2** ([9, Lemma 2.3]). *Let $c \geq 2$ be a sufficiently large constant, and let $D = c \cdot (1 + \sigma\sqrt{n \log n})$. Then $\mathbb{P}(\mathcal{X} \not\subseteq [-D, D]^d) \leq 1/n!$.*

The next lemma is a reformulation of another result by Manthey & Veenstra [9]. The lemma is very useful in conjunction with Lemma 4, as we will have cause to condition on the outcome of drawing noncentral d-dimensional chi random variables.

▶ **Lemma 3** ([9, Lemma 2.8]). *The noncentral d-dimensional chi distribution with parameter $\mu > 0$ and standard deviation σ stochastically dominates the central d-dimensional chi distribution with the same standard deviation.*

The following lemma from Manthey & Veenstra is slightly generalized compared to its original statement. We do not provide a proof, since the original proof remains valid when simply replacing the original assumption with ours.

▶ **Lemma 4** ([9, Lemma 2.7]). *Assume $c \in \mathbb{R}_{\geq 0}$ is a fixed constant and $d \in \mathbb{N}$ is fixed and arbitrary with $d > c$. Let χ_d denote the d-dimensional chi distribution with variance σ^2. Then*

$$\int_0^{\infty} \chi_d(x) x^{-c} \, \mathrm{d}x = \Theta\left(\frac{1}{d^{c/2}\sigma^c}\right).$$

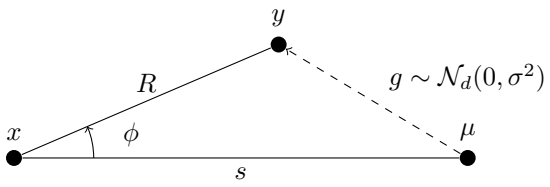


Figure 1 The setting of Theorem 5. As mentioned in the proof of Theorem 5, we may assume without loss of generality that μ lies on L.

2.4.3 Limiting the Adversary

In our analysis we will closely study the angles between edges in the smoothed TSP instance. These angles can be initially specified to our detriment by the adversary. However, the power of the adversary is limited by the strength of the Gaussian perturbations. We quantify the power of the adversary in Theorem 5. See Figure 1 for a sketch accompanying the theorem.

▶ **Theorem 5.** *Let L be some line in $\mathbb{R}^d$, and let $x \in L$. Let y be a point drawn from a d-dimensional Gaussian distribution with mean $\mu \in \mathbb{R}^d$ and variance σ^2. Let ϕ denote the angle between L and $x - y$, and let $R = \|x - y\|$ and $s = \|x - \mu\|$. Let $f_{\phi|R=r}$ denote the density of ϕ, conditioned on a specific outcome $r > 0$ for R. Then for all $d \geq 2$,*

$$\sup_{\phi \in [0,\pi]} f_{\phi|R=r}(\phi) = O\left(\sqrt{d} + \frac{\sqrt{rs}}{\sigma}\right).$$

Moreover, for $d \geq 3$,

$$\sup_{\phi \in (0,\pi)} \frac{f_{\phi|R=r}(\phi)}{\sin\phi} = O\left(\sqrt{d} + \frac{rs}{\sigma^2\sqrt{d}}\right).$$

Theorem 5 yields the following corollary, which provides information on the angle between two Gaussian random points in $\mathbb{R}^d$ with respect to some third point.

▶ **Corollary 6.** *Let $x \in \mathbb{R}^d$. Let y and z be drawn from d-dimensional Gaussian distributions with arbitrary means and the same variance σ^2. Let ϕ denote the angle between $y - x$ and $z - x$, and let $R = \|x - y\|$ and $S = \|x - z\|$. Let $f_{\phi|R=r,S=s}$ denote the density of ϕ conditioned on some outcome $r > 0$ for R and $s > 0$ for S. Then for all $d \geq 2$,*

$$\sup_{\phi \in [0,\pi]} f_{\phi|R=r,S=s}(\phi) = O\left(\sqrt{d} + \frac{\sqrt{\min\{r\bar{r}, s\bar{s}\}}}{\sigma}\right),$$

where $\bar{r} = \|x - \mathbb{E}(y)\|$ and $\bar{s} = \|x - \mathbb{E}(z)\|$. Moreover, for $d \geq 3$,

$$\sup_{\phi \in (0,\pi)} \frac{f_{\phi|R=r,S=s}(\phi)}{\sin\phi} = O\left(\sqrt{d} + \frac{\min\{r\bar{r}, s\bar{s}\}}{\sigma^2\sqrt{d}}\right).$$

3 Analysis of Single 2-Changes

To improve upon the previous analyses, it pays to examine where the analysis of Euclidean 2-opt with Gaussian perturbations [9] fails for $d \in \{2, 3\}$. The problem is that in the course of the proof, Manthey & Veenstra compute

$$\int_0^\infty \frac{1}{x^2} \chi_{d-1}(x)\,\mathrm{d}x,$$

where χ_d denotes the d-dimensional chi distribution. This integral is finite only when $d \geq 4$.

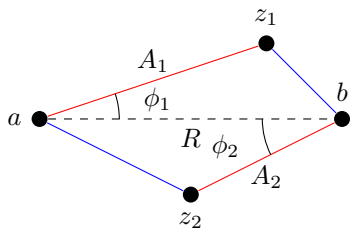


Figure 2 Labels of points and angles involved in a single 2-change.

This problem does not appear in the results obtained by Englert et al. [4]. They consider a more general model of smoothed analysis wherein the adversary specifies a probability density for each point in the TSP instance independently. Since the only information available on the probability densities is their upper bound, they consider a simplified model of a 2-change to keep the analysis tractable. The analysis is then translated to their generic model, which incurs a factor which is super-exponential in d.

Even when one considers d to be a constant as Englert et al. do, the genericity of their model still comes at a cost when translated to a smoothed analysis with Gaussian perturbations, eventually yielding a bound which is polynomial in σ^{-d}.

Specifying the perturbations as Gaussian enables us to analyze the true random experiment modeling a 2-change more closely, as we know the distributions of the distances between points in the smoothed instance. Combined with Theorem 5, which provides information on the angles between edges in the instance, we can carry out an analysis that improves on both Englert et al.'s as well as Manthey & Veenstra's analysis.

We first set up our model of a 2-change perturbed by Gaussian perturbations. To obtain a bound for this case, we first formulate a different analysis of single 2-changes. Consider a 2-change involving the points $\{a, b, z_1, z_2\} \subseteq [-D, D]^d$, where the edges $\{a, z_1\}$ and $\{b, z_2\}$ are replaced by $\{b, z_1\}$ and $\{a, z_2\}$. The improvement to the tour length due to this 2-change is

$$\Delta = \|a - z_1\| - \|b - z_1\| + \|b - z_2\| - \|a - z_2\|.$$

To analyze Δ, we first define $A_1 := \|a - z_1\|$, $A_2 := \|b - z_2\|$, and $R := \|a - b\|$. Moreover, we identify the angle ϕ_1 as the angle between $a - z_1$ and $a - b$, and restrict it to $[0, \pi]$. The corresponding angle ϕ_2 is defined similarly. The restriction of these angles to $[0, \pi]$ is without loss of generality; one may readily observe from Figure 2 that flipping the sign of either ϕ_1 or ϕ_2 does not change the value of Δ.

While Figure 2 may give the impression that we are restricting the analysis to the $d = 2$ case, the analysis is valid for any $d \geq 2$. The two triangles $\triangle az_1b$ and $\triangle az_2b$ will lie in two separate planes in general. The distances involved must thus be understood as d-dimensional Euclidean distances.

With these definitions, we have $\Delta = \eta_1 + \eta_2$, where for $i \in [2]$

$$\eta_i = A_i - \sqrt{A_i^2 + R^2 - 2A_iR\cos\phi_i},$$

which follows from the Law of Cosines.

Suppose we condition on the events $A_1 = a_1$, $A_2 = a_2$, and $R = r$, for some $a_1, a_2, r > 0$. Under these events, η_1 and η_2 are independent random variables. Moreover, Δ is completely fixed by revealing the angles ϕ_1 and ϕ_2. Since we condition on $A_i = a_i$, we can then bound the density of ϕ_i using Corollary 6.

We can use this independence to obtain bounds for $\mathbb{P}(\Delta \in (0, \epsilon])$ for some small $\epsilon > 0$ under these events, for various orderings of a_1, a_2 and r. These bounds are given in Lemma 10.

We begin by obtaining a bound to the density of η_i, $i \in [2]$, using the fact that all randomness in η_i is contained in the angle ϕ_i under the conditioning that $A_i = a_i$ and $R = r$. We denote by $f_{\phi_i|R=r,A_i=a_i}$ the density of the angle ϕ_i, conditioned on $R = r$ and $A_i = a_i$.

▶ **Lemma 7.** *Let $i \in [2]$. The density of $\eta_i = \|a - z_i\| - \|b - z_i\|$, conditioned on $A_i = a_i$ and $R = r$, is bounded from above by*

$$\frac{a_i + r}{a_i r} \cdot \frac{f_{\phi_i|R=r,A_i=a_i}(\phi_i(\eta))}{|\sin \phi_i(\eta)|},$$

where $\phi_i(\eta) = \arccos\left(\frac{a_i^2+r^2-(a_i-\eta)^2}{2a_i r}\right)$.

Proof. Let the conditional density of η_i be $f_{\eta_i|R=r,A_i=a_i}$. Since ϕ_i is restricted to $[0, \pi]$ by assumption, there exists a bijection between η_i and ϕ_i. To be precise, we have

$$\phi_i(\eta_i) = \arccos\left(\frac{a_i^2 + r^2 - (a_i - \eta_i)^2}{2a_i r}\right).$$

By standard transformation rules of probability densities, it holds that

$$f_{\eta_i|R=r,A=a_i}(\eta) = \left|\frac{\mathrm{d}\phi_i(\eta)}{\mathrm{d}\eta}\right| f_{\phi_i|R=r,A_i=a_i}(\phi_i(\eta)).$$

The derivative is easily evaluated:

$$\frac{\mathrm{d}\phi_i(\eta)}{\mathrm{d}\eta} = \frac{-1}{\sqrt{1 - \left(\frac{a_i^2+r^2-(a_i-\eta)^2}{2a_i r}\right)}} \cdot \frac{a_i - \eta}{a_i r} = \frac{-1}{\sin \phi(\eta)} \cdot \frac{a_i - \eta}{a_i r}.$$

Finally, we have $a_i - \eta \leq a_i + r$, which follows from the triangle inequality. This concludes the proof. ◀

By Corollary 6, we have an upper bound for $f_{\phi_i|R=r,A_i=a_i}$. Unfortunately, simply inserting this upper bound is not enough for us to bound $f_{\eta_i|A_i=a_i,R=r}$, since the density as obtained from Lemma 7 diverges for $\phi = 0$ and $\phi = \pi$. There is however a way to cure this divergence.

We now consider a full 2-change (cf. Figure 2). To analyze the improvement Δ caused by this 2-change, we construct a random experiment, conditioned on the outcomes $A_1 = a_1$, $A_2 = a_2$, and $R = r$. We write this random experiment in Algorithm 1, since we will need to execute different experiments depending on the ordering of the values of a_1, a_2 and r. The parameters b_1 and b_2 of this algorithm will take values in $\{a_1, a_2, r\}$, depending on this ordering.

The function `RandomExpt` outlined in Algorithm 1 branches on the outcome of the variable $Z_i = \sqrt{b_i} \sin \phi_i$, $i \in [2]$, where b_i is some distance; we will choose b_i among $\{r, a_i\}$ in subsequent lemmas.

Note that `RandomExpt` returns a tuple (i, ϕ), where $i \in [2]$. We call the angle returned by `RandomExpt` the *good angle*. Moreover, we label the event $i = 1$ as E_1, and $i = 2$ by E_2. The crux of the analysis is now to analyze η_1 if E_1 occurs, and η_2 if E_2 occurs, as under E_i the density of η_i is bounded from above.

Algorithm 1 The algorithm we use to model a random 2-change with fixed $A_1 = a_1$, $A_2 = a_2$, and $R = r$.

Data: Distances $b_1, b_2 > 0$.
Function `RandomExpt`(b_1, b_2)**:**
  Draw $\phi_1 \sim f_{\phi|R=r,A_1=a_1}$
  Draw $\phi_2 \sim f_{\phi|R=r,A_2=a_2}$
  if $\sqrt{b_1}\sin\phi_1 > \sqrt{b_2}\sin\phi_2$ **then**
    return $(1, \phi_1)$
  else
    return $(2, \phi_2)$
  end

▶ **Lemma 8.** *Let $(i, \phi) =$ `RandomExpt`(b_1, b_2) for some $b_1, b_2 > 0$. Let $j = 3 - i$. The density of ϕ, conditioned on $R = r$, $A_1 = a_1$, $A_2 = a_2$, is then bounded from above by*

$$\frac{2M_{\phi_1}M_{\phi_2}}{\mathbb{P}(E_i)} \cdot \arcsin\left(\min\left\{1, \sqrt{\frac{b_i}{b_j}}\sin\phi\right\}\right),$$

where $M_{\phi_i} = \max_{0\le\phi\le\pi} f_{\phi_i|R=r,A_i=a_i}(\phi)$.

Proof. We omit the conditioning on $A_1 = a_1$, $A_2 = a_2$ and $R = r$ in the following, for the sake of clarity. We prove only the case $i = 1$, thus conditioning on E_1, as the proof for $i = 2$ proceeds essentially identically.

Let $X_i = \sqrt{b_i}\sin\phi_i$, $i \in [2]$. The event E_1 is then equivalent to $X_1 > X_2$. Let Z in turn denote the random variable given by X_1 conditioned on E_1. The cumulative distribution function of Z is equal to

$$F_Z(x) = \mathbb{P}(X_1 \le x \mid X_1 > X_2) = \frac{\mathbb{P}(X_1 \le x \wedge X_1 > X_2)}{\mathbb{P}(E_1)}.$$

By the independence of X_1 and X_2, this is equal to

$$F_Z(x) = \frac{1}{\mathbb{P}(E_1)} \cdot \int_0^x f_{X_1}(y) \int_0^y f_{X_2}(z)\,\mathrm{d}z\,\mathrm{d}y.$$

Computing the density of Z is then simply a matter of differentiation. Since $\mathbb{P}(E_1)$ does not depend on x, we obtain

$$f_Z(x) = \frac{1}{\mathbb{P}(E_1)} \cdot f_{X_1}(x) \int_0^x f_{X_2}(z)\,\mathrm{d}z.$$

We next require the density of $X_i = \sqrt{b_i}\sin\phi_i$. Observe that

$$\mathbb{P}(X_i \le x) = \mathbb{P}\Big(\phi_i \le \arcsin(x/\sqrt{b_i})\Big) + \mathbb{P}\Big(\phi_i \ge \pi - \arcsin(x/\sqrt{b_i})\Big). \tag{1}$$

Differentiating this expression to x, we find for $x < \sqrt{b_i}$

$$\begin{aligned} f_{X_i}(x) &= \frac{\mathrm{d}}{\mathrm{d}x}\Big(\mathbb{P}\Big(\phi_i \le \arcsin(x/\sqrt{b_i})\Big) + 1 - \mathbb{P}\Big(\phi_i \ge \pi - \arcsin(x/\sqrt{b_i})\Big)\Big) \\ &= \frac{\mathrm{d}}{\mathrm{d}x}\left(\arcsin\left(\frac{x}{\sqrt{b_i}}\right)\right) \cdot \left[f_{\phi_i}\left(\arcsin\left(\frac{x}{\sqrt{b_i}}\right)\right) + f_{\phi_i}\left(\pi - \arcsin\left(\frac{x}{\sqrt{b_i}}\right)\right)\right] \\ &= \frac{1}{\sqrt{b_i - x^2}} \cdot \left[f_{\phi_i}\left(\arcsin\left(\frac{x}{\sqrt{b_i}}\right)\right) + f_{\phi_i}\left(\pi - \arcsin\left(\frac{x}{\sqrt{b_i}}\right)\right)\right], \end{aligned}$$

and 0 for $x \geq \sqrt{b_i}$. Letting $M_{\phi_i} = \max_{0 \leq \phi \leq \pi} f_{\phi_i | R=r, A_i=a_i}(\phi)$, which exists by Corollary 6, we obtain

$$f_{X_i}(x) \leq 2M_{\phi_i} \cdot \begin{cases} \frac{1}{\sqrt{b_i - x^2}}, & \text{if } x < \sqrt{b_i}, \\ 0, & \text{otherwise.} \end{cases}$$

Using this density, together with the identity $\int_0^x (\sqrt{b} - y^2)^{-1/2} \, \mathrm{d}y = \arcsin(x/\sqrt{b})$ for $x < \sqrt{b}$, we obtain

$$f_Z(x) \leq \frac{2M_{\phi_1} M_{\phi_2}}{\mathbb{P}(E_1)} \cdot \frac{\arcsin\left(\min\left\{1, \frac{x}{\sqrt{b_2}}\right\}\right)}{\sqrt{b_1 - x^2}}$$

if $x < \sqrt{b_1}$, and $f_Z(x) = 0$ otherwise. It remains to convert Z back to ϕ, where ϕ is the good angle. Since we have conditioned on E_1, we know that $Z = \sqrt{b_1} \sin \phi$. Using similar considerations as used in Equation (1), we have

$$f_Z(x) = \frac{1}{\sqrt{b_1 - x^2}} f_\phi(\arcsin(x/\sqrt{b_1})) + \frac{1}{\sqrt{b_1 - x^2}} f_\phi(\pi - \arcsin(x/\sqrt{b_1})).$$

Since this expression holds for all $x \in (0, \sqrt{b_1})$, and since probability densities are non-negative, it follows that

$$f_\phi(\phi) \leq \frac{2M_{\phi_1} M_{\phi_2}}{\mathbb{P}(E_1)} \cdot \arcsin\left(\min\left\{1, \sqrt{\frac{b_1}{b_2}} \sin \phi\right\}\right),$$

for all $\phi \in (0, \pi)$. ◀

For the next part, we apply Lemma 8 to Lemma 7 to bound the density of η_i, given that E_i occurs.

▶ **Lemma 9.** *Let $i \in [2]$ and $j = 3 - i$. Let $f_{\eta_i | E_i}$ denote the density of η_i, conditioned on E_i as well as the outcomes $R = r$, $A_1 = a_1$, and $A_2 = a_2$. Then*

$$f_{\eta_i | E_i}(\eta) \leq \frac{1}{\mathbb{P}(E_i)} \cdot \frac{2\pi M_{\phi_1} M_{\phi_2}}{\min\{a_1, r\} \min\{a_2, r\}},$$

where $M_{\phi_i} = \max_{0 \leq \phi \leq \pi} f_{\phi_i | R=r, A_i=a_i}(\phi)$.

Proof. We prove only the case $i = 1$. From Lemma 7, we know that

$$f_{\eta_i | E_i}(\eta) \leq \frac{a_i + r}{a_i r} \cdot \frac{f_{\phi_i | E_i, A_1=a_1, A_2=a_2}(\phi)}{\sin \phi}.$$

Let $(i, \phi) = \texttt{RandomExpt}(b_1, b_2)$, for some $b_1, b_2 > 0$. We will choose values for b_1 and b_2 depending on the ordering of a_1, a_2 and r. Note that we may do this, since we know the choices of a_1, a_2 and r before executing `RandomExpt`.

Since we condition on E_1, we know that $i = 1$, and hence that ϕ_1 is the good angle. By Lemma 8, we can obtain a bound for $f_{\phi | E_i, A_1=a_1, A_2=a_2, R=r}$. We thus find

$$f_{\eta_1 | E_1}(\eta) \leq \frac{2M_{\phi_1} M_{\phi_2}}{\mathbb{P}(E_1)} \cdot \frac{a_1 + r}{a_1 r} \cdot \frac{\arcsin\left(\min\left\{1, \sqrt{\frac{b_1}{b_2}} \sin \phi\right\}\right)}{\sin \phi}.$$

First, suppose $\sin\phi \geq \sqrt{b_2/b_1}$. Then the arcsine evaluates to $\pi/2$, and so the above is bounded from above by

$$\frac{\pi}{2}\sqrt{\frac{b_1}{b_2}}.$$

Second, suppose $\sin\phi < \sqrt{b_2/b_1}$. Since $\arcsin(x) \leq \pi x/2$ for $x \in (0,1)$, this case yields the same bound, and we obtain

$$f_{\eta_1|E_1}(\eta) \leq \frac{\pi M_{\phi_1} M_{\phi_2}}{\mathbb{P}(E_1)} \cdot \frac{a_1 + r}{a_1 r} \cdot \sqrt{\frac{b_1}{b_2}}$$

We now examine the four cases in the lemma statement.

Case 1: $a_1, a_2 \leq r$.

We let $b_1 = a_1$ and $b_2 = a_2$. Then we have

$$\frac{a_1 + r}{a_1 r} \cdot \sqrt{\frac{a_1}{a_2}} = \frac{a_1 + r}{r\sqrt{a_1 a_2}} \leq \frac{2r}{r\sqrt{a_1 a_2}} = \frac{2}{\sqrt{a_1 a_2}}.$$

Case 2: $a_1, a_2 \geq r$.

We let $b_1 = b_2 = r$, and obtain

$$\frac{a_1 + r}{a_1 r} \leq \frac{2a_1}{a_1 r} = \frac{2}{r}.$$

Case 3: $a_1 \geq r \geq a_2$.

We let $b_1 = r$ and $b_2 = a_2$, which yields

$$\frac{a_1 + r}{a_1 r} \cdot \sqrt{\frac{r}{a_2}} = \frac{a_1 + r}{\sqrt{a_2 r} a_1} \leq \frac{2}{\sqrt{a_2 r}}.$$

Case 4: $a_2 \geq r \geq a_1$.

We let $b_1 = a_1$ and $b_2 = r$, to find

$$\frac{a_1 + r}{a_1 r}\sqrt{\frac{a_1}{r}} \leq \frac{2r\sqrt{a_1}}{a_1 r\sqrt{r}} = \frac{2}{\sqrt{a_1 r}}.$$

This final case concludes the proof. ◀

The bound on the density of η_i from Lemma 9 puts us in the position to prove a bound on the probability that $\Delta \in (0, \epsilon]$.

▶ **Lemma 10.** *Let Δ denote the improvement of a 2-change. Then*

$$\mathbb{P}(\Delta \in (0,\epsilon] \mid A_1 = a_1, A_2 = a_2, R = r) \leq \frac{\pi M_{\phi_1} M_{\phi_2} \epsilon}{\min\{a_1, r\}\min\{a_2, r\}},$$

where $M_{\phi_i} = \max_{0 \leq \phi \leq \pi} f_{\phi_i | R=r, A_i = a_i}(\phi)$.

Proof. We condition first on E_1, and then let an adversary choose an outcome for η_2, say, $\eta_2 = t$. Then we have $\Delta \in (0, \epsilon]$ iff $\eta_1 \in (-t, -t + \epsilon]$, which is an interval of size ϵ.

Since the probability that η_1 falls into an interval of size ϵ is at most $\epsilon \cdot \max_\eta f_{\eta_1|E_1}(\eta)$, all we need to conclude the proof for E_1 is a bound on $f_{\eta_1|E_1}(\eta)$. This is provided by Lemma 9.

We then repeat the same argument for E_2. The result is obtained by applying the Law of Total Probability. ◀

4 Linked Pairs of 2-Changes

To obtain bounds on the smoothed complexity of 2-opt, we consider so-called linked pairs of 2-changes, introduced previously by Englert et al [4]. A pair of 2-changes is said to be linked if some edge removed from the tour by one 2-change is added to the tour by the other 2-change.

Such linked pairs have been considered in several previous works [4, 9]. In each case, the distinction has been made between several types of linked pairs. In our analysis, only two of these types are relevant, and so we will describe only these types for the sake of brevity.

We consider 2-changes which share exactly one edge, and subdivide them into pairs of type 0 and of type 1. A generic 2-change removes the edges $\{z_1, z_2\}$ and $\{z_3, z_6\}$ while adding $\{z_1, z_6\}$ and $\{z_2, z_3\}$. The other 2-change removes $\{z_3, z_4\}$ and $\{z_5, z_6\}$ while adding $\{z_3, z_6\}$ and $\{z_4, z_5\}$. Note that $\{z_3, z_6\}$ occurs in both 2-changes.

- If $|\{z_1, \dots, z_6\}| = 6$, then we say the linked pair is of type 0.
- If $|\{z_1, \dots, z_6\}| = 5$, then we say the linked pair is of type 1.

Type 1 can itself be subdivided into two types, 1a and 1b. We will detail this distinction in Section 4.2.

Before moving on to analyzing linked pairs, we state a useful lemma that justifies limiting the discussion to just linked pairs of types 0 and 1.

▶ **Lemma 11** ([4, Lemma 9]). *In every sequence of t consecutive 2-changes the number of disjoint pairs of 2-changes of type 0 or type 1 is at least $\Omega(t) - O(n^2)$.*

4.1 Type 0

We begin with type 0, as this is by far the simplest linked pair. For clarity, see Figure 3 (left) for an illustration of a type 0 linked pair. It should be noted that, while Figure 3 shows a specific configuration of vertices in two dimensions, the results of this section hold generally; the analysis does not depend on any point having a particular orientation with respect to its neighbors. The same holds for the results in Section 4.2.

The improvement of a type 0 linked pair is completely specified by a small number of random variables. We require five distances between vertices, $R_1 = \|z_1 - z_3\|$, $A_1 = \|z_3 - z_6\|$, $A_2 = \|z_1 - z_2\|$, $R_2 = \|z_4 - z_6\|$ $A_3 = \|z_4 - z_5\|$. Additionally, we need the following angles:

1. ϕ_1 between $z_2 - z_1$ and $z_3 - z_1$,
2. ϕ_2 between $z_1 - z_3$ and $z_6 - z_3$,
3. ϕ_1' between $z_3 - z_6$ and $z_4 - z_6$,
4. ϕ_3 between $z_6 - z_4$ and $z_5 - z_4$.

Note that, if we condition on $A_1 = a_1$, the events $\Delta_1 \in (0, \epsilon]$ and $\Delta_2 \in (0, \epsilon]$ are independent. We can then apply Lemma 10, together with several applications of Lemma 4.

▶ **Lemma 12.** *Let $\Delta_{\min}^{\text{link}}$ denote the minimum improvement of any type 0 pair of linked 2-changes, and assume that $\mathcal{X} \subseteq [-D, D]^d$. Then*

$$\mathbb{P}(\Delta_{\min}^{\text{link}} \in (0, \epsilon]) = O\left(\frac{dD^2 n^6 \epsilon^2}{\sigma^4}\right).$$

4.2 Type 1

As mentioned previously, type 1 linked pairs can be subdivided into two distinct subtypes. Subtype 1a shares exactly one edge between the two 2-changes, while subtype 1b shares two edges.

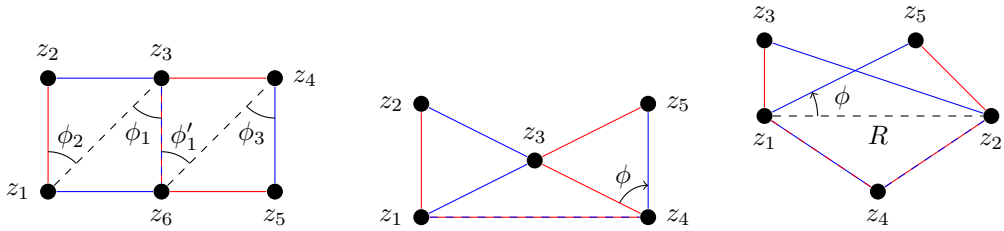


Figure 3 Labels of points involved in the three types of pairs of linked 2-changes. Left: type 0. Center: type 1a. Right: type 1b.

4.2.1 Type 1a

We first consider type 1a. See Figure 3 (center) for a graphical representation of the type, as well as the labels of the points and edges involved.

Let the 2-change replacing $\{z_1, z_2\}$ and $\{z_3, z_4\}$ by $\{z_2, z_3\}$ and $\{z_1, z_4\}$ be called S_1, and the 2-change replacing $\{z_1, z_4\}$ and $\{z_3, z_5\}$ by $\{z_1, z_3\}$ and $\{z_4, z_5\}$ be called S_2.

We proceed by conditioning on $A_2 = \|z_3 - z_4\| = a_2$ and $A_3 = \|z_4 - z_5\| = a_3$. Using Lemma 10, we can then compute the probability that $\Delta_1 \in (0, \epsilon]$. Moreover, the location of z_5 is then still random. Hence, the random variable $\eta = \|z_3 - z_5\| - \|z_4 - z_5\|$ can be analyzed independently from Δ_1.

For the density of η, we have the following lemma from Englert et al [4].

▶ **Lemma 13** ([4, Lemma 15, modified]). *Let $i \in [2]$, and assume that $\mathcal{X} \subseteq [-D, D]^d$. For $a_2, a_3 \in (0, 2\sqrt{d}D]$ and $\eta \in (-a_2, \min\{a_2, 2a_3 - a_2\})$,*

$$f_{\eta | A_2 = a_2, A_3 = a_3}(\eta) \leq M_\phi \cdot \begin{cases} \sqrt{\frac{2}{a_2^2 - \eta^2}}, & \text{if } a_3 \geq a_2, \\ \sqrt{\frac{2}{(a_2+\eta)(2a_3 - a_2 - \eta}}, & \text{if } a_3 < a_2, \end{cases}$$

where $M_\phi = \max_{0 \leq \phi \leq \pi} f_{\phi | A_2 = a_2, A_3 = a_3}(\phi)$. For $\eta \notin (-r, \min\{a_2, 2a_3 - a_2\})$, the density vanishes.

Note that the factor M_ϕ was not present in the original statement of Lemma 13. This is because the original statement concerned a simplified random experiment, wherein the points z_5 and z_3 are chosen uniformly from a hyperball centered on z_4. As such, ϕ is assumed to be distributed uniformly[1]. Since we do not analyze a simplified random experiment, we cannot make this assumption. However, examining the original proof of Lemma 13, this can be resolved by simply inserting the upper bound of the density of ϕ, conditioned on $A_2 = a_2$ and $A_3 = a_3$. This bound is provided to us by Corollary 6.

▶ **Lemma 14.** *Let Δ_2 be the improvement yielded by S_2, and assume that $\mathcal{X} \subseteq [-D, D]^d$. Then*

$$\mathbb{P}(\Delta_2 \in (0, \epsilon] \mid A_2 = a_2) = O\left(\left(\frac{d^{1/4}\sqrt{D}}{\sigma} + \sqrt{\frac{d}{a_2}}\right) \cdot \sqrt{\epsilon}\right).$$

[1] This assumption is only valid for $d = 2$. To see this, observe that by conditioning on $A_i = a_i$, the point z_i is distributed uniformly on the $(d-1)$-sphere with radius a_i. For $d > 2$, the density of ϕ is thus concentrated near $\phi = \pi/2$. An upper bound for this density can be obtained by setting $s = 0$ in Theorem 5, yielding $O(\sqrt{d})$. As Englert et al. assume d to be constant, this has no effect on their eventual result.

Using Lemmas 4 and 14, we can easily prove the following statement about type 1a pairs of 2-changes.

▶ **Lemma 15.** *Let $\Delta_{\min}^{\text{link}}$ denote the minimum improvement of any type 1a pair of 2-changes, and assume that $\mathcal{X} \subseteq [-D, D]^d$. Then*

$$\mathbb{P}(\Delta_{\min}^{\text{link}} \in (0, \epsilon]) = O\left(\frac{n^5 d^{3/4} D^{3/2}}{\sigma^3} \epsilon^{3/2}\right).$$

4.2.2 Type 1b

The final type of linked pair we consider is type 1b. See Figure 3 (right) for a graphical representation.

Let S_1 denote the 2-change replacing $\{z_1, z_3\}$ and $\{z_2, z_4\}$ with $\{z_2, z_3\}$ and $\{z_1, z_4\}$, and let S_2 denote the 2-change replacing $\{z_2, z_5\}$ and $\{z_1, z_4\}$ with $\{z_1, z_5\}$ and $\{z_2, z_5\}$. From Figure 3, it is evident that we can treat Δ_1 and $\eta = \|z_2 - z_5\| - \|z_1 - z_5\|$ as independent variables, as long as we condition on $R = r$.

▶ **Lemma 16.** *Let $\Delta_{\min}^{\text{link}}$ denote the minimum improvement of any type 1b pair of 2-changes, and assume that $\mathcal{X} \subseteq [-D, D]^d$. Then*

$$\mathbb{P}(\Delta_{\min}^{\text{link}} \in (0, \epsilon]) = O\left(\frac{n^5 d^{3/4} D^{3/2}}{\sigma^3} \epsilon^{3/2}\right).$$

Lemmas 12, 15, and 16 enable us to prove an upper bound to the smoothed complexity of 2-opt in the present probabilistic model.

▶ **Theorem 17.** *The expected number of iterations performed by 2-opt for smoothed Euclidean instances of TSP in $d \geq 2$ dimensions is bounded from above by $O\left(dD^2 n^{4+\frac{1}{3}}/\sigma^2\right)$.*

Proof. We assume for this proof that the entire instance is contained within $[-D, D]^d$, with $D = \Theta(1 + \sigma\sqrt{n \log n})$. This occurs with probability at least $1 - 1/n!$. Thus, with probability at least $1 - 1/n!$, the longest tour in the instance has length at most $2\sqrt{d}Dn$. The assumption that the entire instance lies within this hypercube enables us to use Lemmas 12, 15, and 16, which were proved under this assumption.

Let E denote the event that, among all type 0 and type 1 linked pairs of 2-changes, the pair with the smallest improvement is of type 0, and let E^c denote the event that this pair is of type 1a or type 1b. Let the random variable T denote the number of iterations taken by 2-opt to reach a local optimum.

We first compute $\mathbb{E}(T \mid E)$. We apply Lemma 1 with $\alpha = 2$ and $\beta = 2$, which is feasible due to Lemma 12. We then obtain immediately that $\mathbb{E}(T \mid E) = O(dD^2 n^4/\sigma^2)$.

Next, we compute $\mathbb{E}(T \mid E^c)$. In this case, we apply Lemma 1 with $\alpha = 3/2$ and $\beta = 1$ (cf. Lemmas 15 and 16). This yields $\mathbb{E}(T \mid E^c) = O(dD^2 n^{4+\frac{1}{3}}/\sigma^2)$.

Combining the bounds for E and E^c yields the result. ◀

5 Improving the Analysis for $d \geq 3$

The bottleneck in Theorem 17 stems from Lemmas 15 and 16, which bound the probability that any linked pair of type 1a or type 1b improves the tour by at most ϵ. The probability given by these lemmas is proportional to $\epsilon^{3/2}$, which yields an extra factor of $n^{1/3}$ compared to type 0 linked pairs.

For $d \geq 3$, we can improve this to ϵ^2, yielding improved smoothed complexity bounds. The key to this improvement is to use the second part of Corollary 6 to bound the density of η_i as in Lemma 7. This immediately yields the following result on $\eta_i = \|a - z_i\| - \|b - z_i\|$.

▶ **Lemma 18.** *Let $i \in [2]$, and assume that $\mathcal{X} \subseteq [-D, D]^d$. The density of η_i in $d \geq 3$ dimensions, conditioned on $A_i = a_i$ and $R = r$, is bounded from above by*

$$O\left(\frac{a_i + r}{a_i r} \cdot \left(\sqrt{d} + \frac{D \min\{r, a_i\}}{\sigma^2}\right)\right).$$

Proof. We call the desired density $f_{\eta_i | A=a_i, R=r}$. From Lemma 7, we know that

$$f_{\eta_i | A_i=a_i, R=r}(\eta) \leq \frac{a_i + r}{a_i r} \cdot \frac{f_{\phi_i | A_i=a_i, R=r}(\phi_i(\eta))}{|\sin \phi_i(\eta)|}.$$

Since $d \geq 3$, we can use the second part of Corollary 6 to obtain the desired bound, making use of the assumption that all points fall within $[-D, D]^d$. ◀

Lemma 18 enables us to find an improved version of Lemma 10.

▶ **Lemma 19.** *Let Δ denote the improvement of a 2-change in $d \geq 3$ dimensions. Let $i \in [2]$, and assume that $\mathcal{X} \subseteq [-D, D]^d$. Then*

$$\mathbb{P}(\Delta \in (0, \epsilon] \mid A_i = a_i, R = r) = O\left(\left(\frac{\sqrt{d}}{\min\{a_i, r\}} + \frac{D}{\sigma^2}\right) \cdot \epsilon\right).$$

The following lemma now yields the probability that any linked pair of 2-changes improves the tour by at most ϵ. We omit the proof, since it follow easily from Lemma 19 along the same lines as the lemmas in Section 4.

▶ **Lemma 20.** *Let $\Delta^{\text{link}}_{\min}$ denote the minimum improvement of any linked pair of 2-changes of type 0 or type 1 for $d \geq 3$, and assume that $\mathcal{X} \subseteq [-D, D]^d$. Then*

$$\mathbb{P}(\Delta^{\text{link}}_{\min} \in (0, \epsilon]) = O\left(\frac{D^2 n^6 \epsilon^2}{\sigma^4}\right).$$

We then obtain our result for $d \geq 3$.

▶ **Theorem 21.** *The expected number of iterations performed by 2-opt for smoothed Euclidean instances of TSP in $d \geq 3$ dimensions is bounded from above by $O\left(\sqrt{d} D^2 n^4 / \sigma^2\right)$.*

6 Discussion

For convenience, we provide comparisons of the previous smoothed complexity bounds with our bound from Theorem 17 in Tables 1 and 2. These bounds are provided both for small values of σ and for large values, meaning $\sigma = \Omega(1/\sqrt{n \log n})$ and $\sigma = O(1/\sqrt{n \log n})$.

Observe from Tables 1 and 2 that the bound for $d = 2$ has a worse dependence on n compared to the bound for $d \geq 3$. The technical reasons for this difference can be understood from Section 5. A more intuitive explanation for the difference is that our analysis benefits from large angles between edges in the smoothed TSP instance. In $d = 2$, the density of these angles is maximal when they are small, while for $d \geq 3$ it is maximal when the angles are large. In effect, this means that the adversary has less power to specify these angles to our detriment when $d \geq 3$.

Table 1 Previous and current smoothed complexity bounds for Gaussian noise, for $\sigma = O(1/\sqrt{n \log n})$. Note that for $d \geq 4$, the bounds of Englert et al. include a factor c_d which is super-exponential in d.

	Englert, Röglin & Vöcking [4]	Manthey & Veenstra [9]	This paper
$d = 2$	$O\left(n^{4+\frac{1}{3}}/\sigma^{5+\frac{1}{3}} \cdot \log \frac{n}{\sigma}\right)$	–	$O\left(n^{4+\frac{1}{3}}/\sigma^2\right)$
$d = 3$	$O\left(n^{4+\frac{1}{3}}/\sigma^8 \cdot \log \frac{n}{\sigma}\right)$	–	$O\left(n^4/\sigma^2\right)$
$d \geq 4$	$O\left(c_d \cdot n^{4+\frac{1}{3}}/\sigma^{8d/3}\right)$	$O\left(\sqrt{d}n^4/\sigma^4\right)$	$O\left(\sqrt{d}n^4/\sigma^2\right)$

Table 2 Previous and current smoothed complexity bounds for Gaussian noise, for $\sigma = \Omega(1/\sqrt{n \log n})$. Note that for $d \geq 4$, the bounds of Englert et al. include a factor c_d which is super-exponential in d.

	Englert, Röglin & Vöcking [4]	Manthey & Veenstra [9]	This paper
$d = 2$	$O\left(n^7 \log^{3+\frac{2}{3}} n\right)$	–	$O\left(n^{5+\frac{1}{3}} \log n\right)$
$d = 3$	$O\left(n^{8+\frac{1}{3}} \log^5 n\right)$	–	$O\left(n^5 \log n\right)$
$d \geq 4$	$O\left(c_d \cdot n^{4+\frac{1+4d}{3}} \log^{1+\frac{4d}{3}} n\right)$	$O\left(\sqrt{d}n^6 \log^2 n\right)$	$O\left(\sqrt{d}n^5 \log n\right)$

From these tables, the greatest improvement is made for $d = 3$, where we improve by $n^{3+\frac{1}{3}} \log^4 n$ in the large σ case, and by $\sqrt[3]{n} \log(n/\sigma)/\sigma^6$ for small σ. For $d = 2$, the improvement is more modest at $n^{1+\frac{2}{3}} \log^{2+\frac{2}{3}} n$ for large σ and $\log(n/\sigma)/\sigma^{3+\frac{1}{3}}$ for small σ. For $d \geq 4$, we improve by $n \log n$ for large σ, and by σ^{-2} for small σ.

Note that we improve upon previous bounds mainly in the dependence on the perturbation strength. In an intuitive sense, this is most substantial for instances that are weakly perturbed from the adversarial instance, or in other words, that are close to worst case. In addition, the small-σ case is considered more interesting for a smoothed analysis, since small σ model the intuition of smoothed analysis of a small perturbation, while large σ reduce the analysis basically to an average-case analysis In order to improve the explicit dependence on n, which is the same as for Manthey & Veenstra [9], we believe new techniques are necessary.

As a final comment, we note that the techniques we employed in Sections 3 and 5 can also be used to improve and significantly simplify the analysis of the one-step model used by Englert et al [4]. For $d \geq 3$, the improvement amounts to a factor of $n^{1/3}\phi^{1/6} \log(n\phi)$, while for $d = 2$, the improvement is just $\log(n\phi)$, where ϕ denotes the upper bound of the density functions used in the one-step model.

References

1 Emile Aarts and Jan Karel Lenstra, editors. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003. doi:10.2307/j.ctv346t9c.

2 Barun Chandra, Howard Karloff, and Craig Tovey. New Results on the Old k-opt Algorithm for the Traveling Salesman Problem. *SIAM Journal on Computing*, 28(6):1998–2029, January 1999. doi:10.1137/S0097539793251244.

3 Christian Engels and Bodo Manthey. Average-case approximation ratio of the 2-opt algorithm for the TSP. *Operations Research Letters*, 37(2):83–84, March 2009. doi:10.1016/j.orl.2008.12.002.

4 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst Case and Probabilistic Analysis of the 2-Opt Algorithm for the TSP. *Algorithmica*, 68(1):190–264, January 2014. Corrected version: arXiv:2302.06889. doi:10.1007/s00453-013-9801-4.

5 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Smoothed Analysis of the 2-Opt Algorithm for the General TSP. *ACM Transactions on Algorithms*, 13(1):10:1–10:15, September 2016. doi:10.1145/2972953.

6 Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics. Springer-Verlag, Berlin Heidelberg, 2000. doi:10.1007/978-3-662-21708-5.

7 Bodo Manthey. Smoothed Analysis of Local Search. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 285–308. Cambridge University Press, Cambridge, 2021. doi:10.1017/9781108637435.018.

8 Bodo Manthey and Heiko Röglin. Smoothed Analysis: Analysis of Algorithms Beyond Worst Case. *it – Information Technology*, 53(6):280–286, December 2011. doi:10.1524/itit.2011.0654.

9 Bodo Manthey and Rianne Veenstra. Smoothed Analysis of the 2-Opt Heuristic for the TSP: Polynomial Bounds for Gaussian Noise. In Leizhen Cai, Siu-Wing Cheng, and Tak-Wah Lam, editors, *Algorithms and Computation*, Lecture Notes in Computer Science, pages 579–589, Berlin, Heidelberg, 2013. Springer. Full, improved version: Full, improved version: arXiv:2308.00306. doi:10.1007/978-3-642-45030-3_54.

10 Christos H. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, June 1977. doi:10.1016/0304-3975(77)90012-3.

11 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, May 2004. doi:10.1145/990308.990310.

12 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis: An attempt to explain the behavior of algorithms in practice. *Communications of the ACM*, 52(10):76–84, October 2009. doi:10.1145/1562764.1562785.

On the Complexity of the Eigenvalue Deletion Problem

Neeldhara Misra
Indian Institute of Technology, Gandhinagar, India

Harshil Mittal
Indian Institute of Technology, Gandhinagar, India

Saket Saurabh
Institute of Mathematical Sciences, Chennai, India
University of Bergen, Norway

Dhara Thakkar
Indian Institute of Technology, Gandhinagar, India

Abstract

For any fixed positive integer r and a given budget k, the r-Eigenvalue Vertex Deletion (r-EVD) problem asks if a graph G admits a subset S of at most k vertices such that the adjacency matrix of $G \setminus S$ has at most r distinct eigenvalues. The edge deletion, edge addition, and edge editing variants are defined analogously. For $r = 1$, r-EVD is equivalent to the Vertex Cover problem. For $r = 2$, it turns out that r-EVD amounts to removing a subset S of at most k vertices so that $G \setminus S$ is a cluster graph where all connected components have the same size.

We show that r-EVD is NP-complete even on bipartite graphs with maximum degree four for every fixed $r > 2$, and FPT when parameterized by the solution size and the maximum degree of the graph.

We also establish several results for the special case when $r = 2$. For the vertex deletion variant, we show that 2-EVD is NP-complete even on triangle-free and $3d$-regular graphs for any $d \geqslant 2$, and also NP-complete on d-regular graphs for any $d \geqslant 8$. The edge deletion, addition, and editing variants are all NP-complete for $r = 2$. The edge deletion problem admits a polynomial time algorithm if the input is a cluster graph, while – in contrast – the edge addition variant is hard even when the input is a cluster graph. We show that the edge addition variant has a quadratic kernel. The edge deletion and vertex deletion variants admit a single-exponential FPT algorithm when parameterized by the solution size alone.

Our main contribution is to develop the complexity landscape for the problem of modifying a graph with the aim of reducing the number of distinct eigenvalues in the spectrum of its adjacency matrix. It turns out that this captures, apart from Vertex Cover, also a natural variation of the problem of modifying to a cluster graph as a special case, which we believe may be of independent interest.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Graph Modification, Rank Reduction, Eigenvalues

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.53

Related Version *Full Version*: https://arxiv.org/abs/2310.00600

Funding *Neeldhara Misra*: Supported by DST-SERB and IIT Gandhinagar.
Harshil Mittal: Supported by IIT Gandhinagar.
Saket Saurabh: Supported by ERC, the University of Bergen, and IMSc.
Dhara Thakkar: Supported by CSIR-UGC NET JRF Fellowship.

Acknowledgements We thank Daniel Lokshtanov for helpful discussions.

1 Introduction

Graph modification problems are a fundamental class of optimization problems where we have a class of graphs $\mathcal{F}$ that satisfy some property of interest Π, the input is a graph G, and we are interested in a smallest subset of vertices $S \subseteq V(G)$ such that $G \setminus S \in \mathcal{F}$. This is a rather general framework that captures several classical optimization problems as special cases, for instance:

- when $\mathcal{F}$ is the collection of edgeless graphs, then the problem is Vertex Cover;
- when $\mathcal{F}$ is the collection of acyclic graphs, then the problem is Feedback Vertex Set;
- when $\mathcal{F}$ is the collection of bipartite graphs, then the problem is Odd Cycle Traversal;

and so on. It has also been of interest to study modifications other than vertex deletion: the most common alternate modifications considered include edge deletion, edge addition, and edge editing (adding and removing edges). The optimization problems for these operations may be posed analogously.

Meesum, Misra, and Saurabh [9] pose the question of modifying a graph with the goal of reducing the rank of the associated adjacency matrix, which is to say that $\mathcal{F}_{\leqslant r}$ is the class of graphs whose adjacency matrices have rank at most r. We use A_G to denote the adjacency matrix of a graph G, and we use the phrase "spectrum of G" to refer to the (multi-)set of eigenvalues of A_G. Previous works focus separately on the settings of undirected [9] and directed [10] graphs.

In the setting of simple undirected graphs, Meesum, Misra, and Saurabh [9] introduce and study the r-Rank Vertex Deletion, r-Rank Edge Deletion, and r-Rank Editing problems. These problems generalize the classical Vertex Cover problem. They show that all the three problems are NP-complete, and are fixed parameter tractable (FPT) in the standard parameter: in particular, they demonstrate an algorithm with running time $2^{\mathcal{O}(k \log r)} n^{\mathcal{O}(1)}$ for r-Rank Vertex Deletion, and an algorithm for r-Rank Edge Deletion and r-Rank Editing running in time $2^{\mathcal{O}(f(r)\sqrt{k} \log k)} n^{\mathcal{O}(1)}$, where k is the size of the solution sought. The authors also leave the following question open:

"[...] what is complexity of the problem of reducing the number of distinct eigenvalues of a graph by deleting a few vertices or editing a few edges?"

In this paper, we address this question at length, developing an initial picture of the complexity landscape for what we call the r-Eigenvalue Vertex Deletion (r-EVD), r-Eigenvalue Edge Deletion (r-EED), r-Eigenvalue Edge Addition (r-EEA), and r-Eigenvalue Edge Editing (r-EEE) problems. All these problems are defined for an arbitrary but fixed positive integer r.

The problem definitions are the following, where we are given an undirected graph G and a positive integer k as input in all cases:

- **r-EVD.** Is there a set $S \subseteq V(G)$ of size $\leqslant k$ such that the number of distinct eigenvalues of $A_{G \setminus S}$ is at most r?
- **r-EEE.** Is there a set $F \subseteq \binom{V(G)}{2}$ of size $\leqslant k$ such that the number of distinct eigenvalues of A_H is at most r, where $H := (V(G), E(G) \Delta F)$?
- **r-EEA.** Is there a set $F \subseteq \binom{V(G)}{2} \setminus E(G)$ of size $\leqslant k$ such that the number of distinct eigenvalues of A_H is at most r, where $H := (V(G), E(G) \cup F)$?
- **r-EED.** Is there a set $F \subseteq E(G)$ of size $\leqslant k$ such that the number of distinct eigenvalues of A_H is at most r, where $H := (V(G), E(G) \setminus F)$?

Note that if we have a solution S for the r-Rank Vertex Deletion problem, then S is also a solution for the $(r+1)$-Eigenvalue Vertex Deletion problem; and analogous statements hold for the other modification problems. This is because the r-Rank Vertex Deletion problem can be equivalently stated as follows: given a graph G and a positive

Table 1 A summary of our results. The result marked † holds for all d except for $d = 1, 2, 3, 4, 5, 7$. Some polynomial cases are omitted from this summary.

	$r = 2$	Fixed $r \geqslant 3$
Vertex Deletion	NP-complete for d-regular graphs † (Theorem 5)	NP-complete even on bipartite graphs (Theorem 8)
	FPT in k (Theorem 6)	FPT in k and $\Delta(G)$ (Theorem 9)
	Polynomial time on forests (Proposition 7)	
Edge Addition	NP-complete even on cluster graphs (Theorem 10)	NP-complete (Theorem 12)
	Quadratic kernel in k (Theorem 11)	
Edge Deletion	NP-complete (Theorem 16)	NP-complete (Theorem 15)
	FPT in k (Theorem 13)	
	Polynomial time on triangle-free graphs (Proposition 14)	
Edge Editing	NP-complete (Theorem 16)	OPEN

integer k, find a smallest subset of vertices $S \subseteq V(G)$ such that $G \setminus S$ has at most r non-zero eigenvalues. However, the converse is not true (since, in general, bounding the number of distinct eigenvalues is not sufficient to bound the rank), making the eigenvalue deletion problems distinct from their rank deletion counterpart.

Our Contributions. We summarize our contributions below, and also in Table 1. We first focus on the special case when $r = 2$. It is known that the adjacency matrix A_G of a graph G has at most two distinct eigenvalues if and only if G is a disjoint union of equal-sized cliques (Lemma 1). Based on this, note that the 2-Eigenvalue Vertex Deletion problem is equivalent to finding a subset $S \subseteq V(G)$ of vertices such that $G \setminus S$ is a disjoint union of cliques of size ℓ for some $1 \leqslant \ell \leqslant |V(G)|$. Note that this is closely related to the Cluster Vertex Deletion problem, which is a well-studied question that involves removing a smallest subset of vertices to obtain a cluster graph. However, to the best of our knowledge, the variant where we demand that the clusters have the same size has not been studied. Our results about the "uniform" version of Cluster Vertex Deletion may therefore be of independent interest.

Our main contributions in the context of vertex deletion are the following results:

- We show that 2-EVD is NP-complete on d-regular graphs for all d except for $d = 1, 2, 3, 4, 5, 7$ (Theorem 5).
- We also give a single-exponential FPT algorithm in the standard parameter (Theorem 6), and show that the problem can be solved in polynomial time on forests and d-regular graphs for $d \leqslant 2$ (Proposition 7).
- Further, for any fixed $r \geqslant 3$, we show that r-EVD is NP-complete on bipartite graphs (Theorem 8) and is FPT in the standard parameter combined with the maximum degree of the graph (Theorem 9).

We now describe our findings for the edge modification variants.

- We show that 2-EEA is already NP-complete when the input is either a cluster graph, a forest, or a collection of cycles (Theorem 10).
- We demonstrate that the problem has a quadratic kernel in the standard parameter (Theorem 11).
- We show that r-EEA is NP-complete for any fixed $r \geqslant 3$ (Theorem 12).
- For the edge deletion variant, we show that r-EED is NP-complete for any fixed $r \geqslant 2$ (Theorems 15 and 16).
- For 2-EED, we have a single-exponential FPT algorithm (Theorem 13) in the standard parameter and a polynomial time algorithm on triangle-free graphs (Proposition 14).
- Finally, for the edge editing variant, we show that 2-Eigenvalue Edge Editing is NP-complete (Theorem 16).

Related Work. As we noted previously, the special case when $r = 2$ is closely related to the problem of modifying to a cluster graph, in which we are allowed to modify the graph such that the resulting graph is cluster i.e., it is disjoint union of cliques. Depending on the modifications allowed, these problems are variously refered to as Cluster Vertex Deletion, Cluster Edge Deletion, Cluster Edge Addition and Cluster Edge Editing. Further, Shamir, Sharan, and Tsur [12] have studied a variant of cluster vertex deletion where they additionally demand that the cluster graph obtained after the modification has at most p components.

Problems related to modifying to a cluster graph are very well-studied because they model the clustering problem in various ways. In a clustering problem we are given various data points with some notion of distance between these points, and it is of interest to group these points into "clusters", where each cluster consists of points that are mutually close with respect to the given distance metric. These scenarios can often be modeled with graphs, and in fact graph structure can often be used to model additional constraints of interest. Given the fundamental importance of clustering, it is no surprise that modifying to cluster graphs has attracted substantial interest in the literature of graph algorithms. We refer the reader to [11] for an overview of results related to cluster modification problems.

Another related problem is the problem of deleting to a graph where the connected components have small diameter. This is known as the s-Club Cluster Vertex Deletion problem [3]. Here, we are given a graph G and two integers $s \geqslant 2$ and $k \geqslant 1$; and the question is if it is possible to remove at most k vertices from G such that each connected component of the resulting graph has diameter at most s. Note that this naturally generalizes the problem of modifying to cluster graphs: indeed, the problem is equivalent to Cluster Vertex Deletion for $s = 1$. The edge modification variants have also been considered and are well-studied.

We note that a solution to the r-Eigenvalue Vertex Deletion problem will also be a valid solution to the $(r-1)$-Club Cluster Vertex Deletion due to Lemma 2, which states that graphs of diameter d have at least $(d+1)$ distinct eigenvalues. This is analogously true for the other modification problems as well. On the other hand, it is easy to see that the converse is not necessarily true.

Remarks. Due to lack of space, we describe most proofs informally and defer a detailed exposition to a full version of the paper. Such results are marked with a $(\star)$. The full version also has a list of problem definitions and extended technical preliminaries. Throughout, we use the $\mathcal{O}^\star(\cdot)$ notation to suppress polynomial factors.

Sections 3,4,5, and 6 focus respectively on the problems of r-EVD, r-EEA, r-EED, and r-EEE.

2 Preliminaries

Let $G = (V, E)$ be a simple undirected graph, where, $V(G)$ is the vertex set of G and $E(G)$ is the edge set of G. We typically use n and m to denote $|V(G)|$ and $|E(G)|$, respectively. The adjacency matrix $A_G = a_{ij}$ of a graph G is an $n \times n$ matrix with $a_{ij} \in \{0, 1\}$ the entry $(i.j) = 1$ if the pair (i, j) is an edge in G. We note that the spectrum of A_G can be computed in polynomial time. A *principal submatrix* of a square matrix A is a matrix obtained by removing an equal number of rows and columns from A such that the indices of the removed rows match with the indices of the removed columns.

The following known results will be relevant to our discussions:

▶ **Lemma 1** ([8, 5]). *Let G be a graph. Then, its adjacency matrix A_G has at most two distinct eigenvalues if and only if G is a disjoint union of equal-sized cliques.*

▶ **Lemma 2** ([2], Proposition 1.3.3). *Let G be a connected graph with diameter d. Then, its adjacency matrix A_G has at least $d + 1$ distinct eigenvalues.*

▶ **Lemma 3** (Cauchy interlacing; [2], Corollary 2.5.2). *Let A be a symmetric matrix of size $n \times n$. Let B be a principal submatrix of A of size $(n-1) \times (n-1)$. Then, the eigenvalues of B interlace the eigenvalues of A. That is,*

$$\mu_1 \geqslant \sigma_1 \geqslant \mu_2 \geqslant \sigma_2 \geqslant \mu_3 \geqslant \ldots\ldots\ldots\ldots \geqslant \mu_{n-2} \geqslant \sigma_{n-2} \geqslant \mu_{n-1} \geqslant \sigma_{n-1} \geqslant \mu_n$$

where, $\mu_1 \geqslant \mu_2 \geqslant \ldots\ldots \geqslant \mu_n$ denote the n eigenvalues of A, and $\sigma_1 \geqslant \sigma_2 \geqslant \ldots\ldots \geqslant \sigma_{n-1}$ denote the $n-1$ eigenvalues of B.

▶ **Lemma 4** ([2], Chapter 3, Exercise 1). *Let G be a graph with smallest eigenvalue -1. Then, G is a disjoint union of cliques.*

Some examples of graph classes whose spectrum is well-known include complete graphs, paths and cycles ([2], Chapter 1). A complete graph on n vertices has eigenvalues -1 and $n-1$ (with multiplicities $n-1$ and 1 respectively). A path on n vertices has eigenvalues $2\cos\left(\frac{\pi j}{n+1}\right)\big|_{1 \leqslant j \leqslant n}$. A cycle on n vertices has eigenvalues $2\cos\left(\frac{2\pi j}{n}\right)\big|_{0 \leqslant j \leqslant n-1}$.

3 Reducing eigenvalues by deleting vertices

In this section, we show that the r-EVD problem is NP-complete for $r \geqslant 1$. Recall that for $r = 1$, r-EVD is equivalent to Vertex Cover. For $r = 2$, we show that the problem is NP-complete on general graphs, admits a single-exponential FPT algorithm in the standard parameter, and is polynomial-time solvable on trees. For any fixed $r \geqslant 3$, we show that the problem is NP-complete on bipartite graphs and is FPT in the standard parameter combined with the maximum degree of the graph.

3.1 Deleting to Two Distinct Eigenvalues

Note that by Lemma 1, 2-EVD is equivalent to Uniform Cluster Vertex Deletion, a problem where the input is a graph G and a positive integer k and the question is if there is a subset $S \subseteq V(G)$ of vertices such that $G \setminus S$ is a disjoint union of ℓ-sized cliques for some $1 \leqslant \ell \leqslant |V(G)|$. Note that ℓ is not a part of the input. We begin by showing that the problem is hard even when restricted to d-regular graphs for any d other than $1, 2, 3, 4, 5, 7$.

▶ **Theorem 5** ($\star$). *2-Eigenvalue Vertex Deletion is NP-complete even on triangle-free and $3d$-regular graphs for any $d \geqslant 2$, and NP-complete on d-regular graphs for any $d \geqslant 8$.*

To show this result we use two reductions: one from the Independent Set problem on cubic triangle-free graphs and the other from Independent Set on planar cubic triangle-free graphs.

In the first construction, we replace every vertex v with vertices $v^{(1)}$ and $v^{(2)}$, and extended the edges as follows: an edge (u, v) maps to the edges $(u^{(1)}, v^{(1)})$, $(u^{(1)}, v^{(2)})$, $(u^{(2)}, v^{(1)})$, and $(u^{(2)}, v^{(2)})$. Note that this construction preserves triangle-freeness and transforms a cubic graph to a six-regular graph. For demonstrating hardness on $3d$ regular graphs for $d \geqslant 2$, we make d copies of the vertices instead of two copies.

For the second construction, we make six copies of the graph and for every vertex, we induce a clique on all its copies. This construction turns a cubic graph into a 8-regular graph. For demonstrating hardness on d regular graphs for $d \geqslant 8$, we make $(d-2)$ copies of the vertices instead of six.

Next, we note that 2-EVD admits a branch-and-bound-based FPT algorithm that is similar in spirit to the naive branching algorithm for Cluster Vertex Deletion. As long as our instance has an induced path on three vertices $\{u, v, w\}$, we recursively solve the instances $(G \setminus \{u\}, k-1)$, $(G \setminus \{v\}, k-1)$ and $(G \setminus \{w\}, k-1)$. Note that this branching algorithm enumerates all minimal subsets S of size at most k such that $G \setminus S$ is a disjoint union of cliques. At a leaf of any successful execution path of this branching algorithm, we are left with a subgraph H of G that is a cluster graph, and a (possibly reduced) budget $k' \leqslant k$. At this point, we guess the value of ℓ, and extend our solution greedily by: (a) deleting all cliques smaller than ℓ, and (b) for any cliques of size, say q where $q > \ell$, we delete an arbitrary subset of $(q - \ell)$ vertices. We have a valid solution at this if and only if there is some ℓ for which the cost of "uniformizing" the cluster graph H to cliques of size ℓ is within the remaining budget k'.

▶ **Theorem 6.** 2-*Eigenvalue Vertex Deletion can be solved in time* $\mathcal{O}^\star(3^k)$.

Proof. Let us describe a recursive branching algorithm. Consider an instance, say (G, k), of 2-Eigenvalue vertex Deletion. By Lemma 1, our goal is to decide whether we can delete at most k vertices from G to get a disjoint union of equal-sized cliques. First, we check if G has an induced path on three vertices. This takes polynomial time.

Case 1: G has no induced path on three vertices. The graph G is a disjoint union of cliques, say $C_1, \ldots, C_t$, of sizes $s_1, \ldots, s_t$ respectively. We know that deleting the vertices of any solution results in a disjoint union of equal-sized cliques $\big($say, of size $x\big)$. Observe that for each $1 \leqslant i \leqslant t$,

- If $s_i \geqslant x$, then $s_i - x$ vertices of the clique C_i are deleted, leaving behind x of its vertices.
- If $s_i < x$, then the entire clique C_i, i.e., all its s_i vertices, are deleted.

So, the overall solution size, i.e., total number of deleted vertices, is

$$\sum_{\substack{1 \leqslant i \leqslant t: \\ s_i \geqslant x}} (s_i - x) + \sum_{\substack{1 \leqslant i \leqslant t: \\ s_i < x}} s_i = \sum_{i=1}^{t} s_i - x \cdot \mu(x)$$

where $\mu(x)$ denotes the number of s_i's amongst $s_1, \ldots, s_t$ such that $s_i \geqslant x$.

Thus, the size of any minimum-sized solution is

$$\sum_{i=1}^{t} s_i - \max_{1 \leqslant j \leqslant t} \big(s_j \cdot \mu(s_j)\big)$$

If this size is $\leqslant k$, we return **YES**; otherwise, we return **NO**. This takes polynomial time.

Case 2: G has an induced path on three vertices, say $a - b - c$. Note that any solution must pick at least one of its three vertices, i.e., a, b, c. So, if $k = 0$, we return NO; otherwise, we guess a vertex that is picked into solution. That is, we branch as follows: In the first (resp. second and third) branch, we include the vertex a (resp. b and c) into solution, delete it from G, and reduce the parameter k by 1. It takes polynomial time to create the subproblems $(G \setminus \{a\}, k-1)$, $(G \setminus \{b\}, k-1)$ and $(G \setminus \{c\}, k-1)$. Next, we run our algorithm on these three instances. If at least one of these three recursive calls returns **YES**, so do we; otherwise, we return **NO**.

The depth of our search tree is at most k. Also, each of its internal nodes has three children. Therefore, it has at most $\mathcal{O}(3^k)$ nodes. Thus, as we spend polynomial time at each node, the overall running time is at most $\mathcal{O}^\star(3^k)$. ◀

We now show that 2-Eigenvalue Vertex Deletion can be solved in polynomial time when the input is a forest. Let (G, k) be an instance of 2-EVD where G is a forest. Note that if S is a valid solution, then $G \setminus S$ is either independent or a disjoint collection of edges.

Therefore, we can arrive at an optimal solution by computing the size of a maximum independent set and a maximum induced matching: this can be done in polynomial time on forests [1, 13]. We also note that a similar argument applies to d-regular graphs for $d \leqslant 2$.

▶ **Proposition 7** ($\star$). *2-Eigenvalue Vertex Deletion admits polynomial time algorithms on forests and d-regular graphs for $d \leqslant 2$.*

3.2 r-EVD for $r \geqslant 3$

To demonstrate the hardness of r-EVD for any fixed $r \geqslant 3$, we give a reduction from Vertex Cover on Cubic Graphs.

▶ **Theorem 8** ($\star$). *Let $r \geqslant 3$ be an integer. Then, r-Eigenvalue Vertex Deletion is NP-complete, even on bipartite graphs of maximum degree four.*

Next, we show that r-EVD is FPT in the combined parmeter $k + \Delta(G)$, where $\Delta(G)$ is the maximum degree of G.

▶ **Theorem 9.** *Let $r \geqslant 3$ be an integer. Then, r-Eigenvalue Vertex Deletion admits an FPT algorithm running in time $\mathcal{O}^\star\left((r+1)^{2k} \cdot 2^{k^2} \cdot (\Delta(G))^{rk}\right)$.*

Proof Sketch. Let (G, k) be an instance of r-EVD. We claim that if G has more than $(r+1) \cdot 2^k$ eigenvalues, then G is a NO instance, and we can detect this upfront. The intuition is that the Cauchy interlacing structure (Lemma 3) allows us to conclude that one vertex can reduce the number of distinct eigenvalues in the spectrum by a factor of at most half: so if there are "too many" distinct eigenvalues in the spectrum to begin with, k deletions will not suffice to reduce the number of distinct eigenvalues substantially enough. We now quantify this argument: suppose, for the sake of contradiction, that G has more than $(r+1) \cdot 2^k$ eigenvalues, and let $S \subseteq V(G)$ be a subset of at most k vertices such that $A_{G \setminus S}$ has at most r distinct eigenvalues. Denote the vertices of S by $v_1, v_2, \ldots, v_t$, where $t \leqslant k$. By Lemma 3 applied to G, $G \setminus \{v_1\}$, we know that the number of distinct eigenvalues in $G \setminus \{v_1\}$ is at least $\lfloor \frac{1}{2}\eta_G \rfloor$, where η_G is the number of distinct eigenvalues in G. Applying this argument iteratively to $G \setminus \{v_1\}$ and $G \setminus \{v_1, v_2\}$ and so on, it is clear that the number of distinct eigenvalues in $G \setminus S$ is at least $\frac{\eta_G}{2^k} - 1$, but if $\eta_G > (r+1) \cdot 2^k$, then we have a contradiction.

So we assume that G has at most $(r+1)\cdot 2^k$ eigenvalues in its spectrum. Note that if G has a shortest path P with at least r edges then any solution S must contain one of the vertices of P (c.f. Lemma 2). This gives us a branching strategy that can be executed in $\mathcal{O}^\star((r+1)^k)$ time. Let (H, k') be an instance at a leaf of some successful execution path of this branching algorithm. Note that H is a subgraph of G whose diameter is at most $r-1$ and $k' \leqslant k$ is a residual budget.

Let C be a connected component of H. Note that $|C| \leqslant \big(\Delta(G)\big)^r$, in other words, H is a collection of "small" components. Note that if the spectrum of H has more than $(r+1)\cdot 2^{k'}$ eigenvalues, we say NO as before. On the other hand, if the spectrum of H has at most r eigenvalues, then we are already done. So the spectrum of H has more than r and at most $(r+1)\cdot 2^{k'}$ eigenvalues. Otherwise, for the sake of analysis, assume that (H, k') is a **YES**-instance with solution S. Note that there is an eigenvalue λ that belongs to the spectrum of H but not to the spectrum of $H \setminus S$. Note that there is at least one connected component C such that λ belongs to the spectrum of $H[C]$. Therefore, $S \cap C \neq \emptyset$. Our algorithm proceeds by guessing λ and a choice of vertex from $S \cap C$, both of which we can afford because the spectrum of H and the sizes of the components of H are bounded by $(r+1)\cdot 2^{k'}$ and $|C| \leqslant \big(\Delta(G)\big)^r$ respectively. ◀

4 Reducing eigenvalues by adding edges

We show that the 2-Eigenvalue Edge Addition is NP-complete even on cluster graphs, and demonstrate a quadratic kernel in the standard parameter. Also, for any fixed $r \geqslant 3$, we show that the r-EEA problem is NP-complete.

For the first result, we reduce from 3-Partition which is known to be strongly NP-complete [7]. The input for 3-Partition consists of a set $T = \{s_1, \ldots, s_{3n}\}$ and b, where s_i's are positive integers from $\left(\frac{b}{4}, \frac{b}{2}\right)$, s_i's and b are given in unary, and $\sum_{i=1}^{3n} s_i = nb$. The goal of this problem is to decide whether there T can be partitioned into n triplets such that the elements of any triplet sum up to b. The intuition for the reduction is the following: the reduced instance is a disjoint union of cliques whose sizes are $\{s_1, \ldots, s_{3n}\}$ and a large number of cliques of size b. The idea is that a solution to the 3-Partition instance can guide the smaller cliques into appropriate mergers so that all cliques have size b, and the "large" number of cliques of size b, combined with an appropriately chosen budget, essentially forces this solution structure in the reverse direction, allowing us to derive a solution for 3-Partition.

▶ **Theorem 10.** *2-Eigenvalue Edge Addition is NP-complete, even when restricted to cluster graphs, forests, and 2-regular graphs.*

Proof. We describe the hardness for cluster graphs. Consider an instance, say (T, b), of 3-Partition, where $T = \{s_1, \ldots, s_{3n}\}$ such that i) $\frac{b}{4} < s_i < \frac{b}{2}$ for all $1 \leqslant i \leqslant 3n$, and ii) $\sum_{i=1}^{3n} s_i = nb$.

Let us construct a graph, say G, as follows: For every $1 \leqslant i \leqslant 3n$, introduce a clique, say C_i, of size s_i. Also, add $M := 3nb$ cliques, each of size b; let us refer to them as *dummy cliques*. The graph G is the disjoint union of $3n + M$ cliques, namely $C_1, \ldots, C_{3n}$ and the M dummy cliques. Let us show that (T, b) is a **YES** instance of 3-Partition if and only if (G, nb^2) is a **YES** instance of 2-Eigenvalue Edge Addition.

($\Rightarrow$) Suppose that (T, b) is a **YES** instance of 3-Partition. Then, there exists a partition of T into n triplets, say $T = T_1 \uplus \ldots \uplus T_n$, such that for every $1 \leqslant i \leqslant n$, the elements of T_i add up to b. That is, $s_{x_i} + s_{y_i} + s_{z_i} = b$, where $s_{x_i}, s_{y_i}, s_{z_i}$ denote the three elements of T_i.

For every $1 \leqslant i \leqslant n$, merge the three cliques $C_{x_i}, C_{y_i}, C_{z_i}$ into one clique, say D_i, as follows:

- Make every vertex of C_{x_i} adjacent to every vertex of C_{y_i}.
- Make every vertex of C_{x_i} adjacent to every vertex of C_{z_i}.
- Make every vertex of C_{y_i} adjacent to every vertex of C_{z_i}.

See Figure 1 for an illustration.

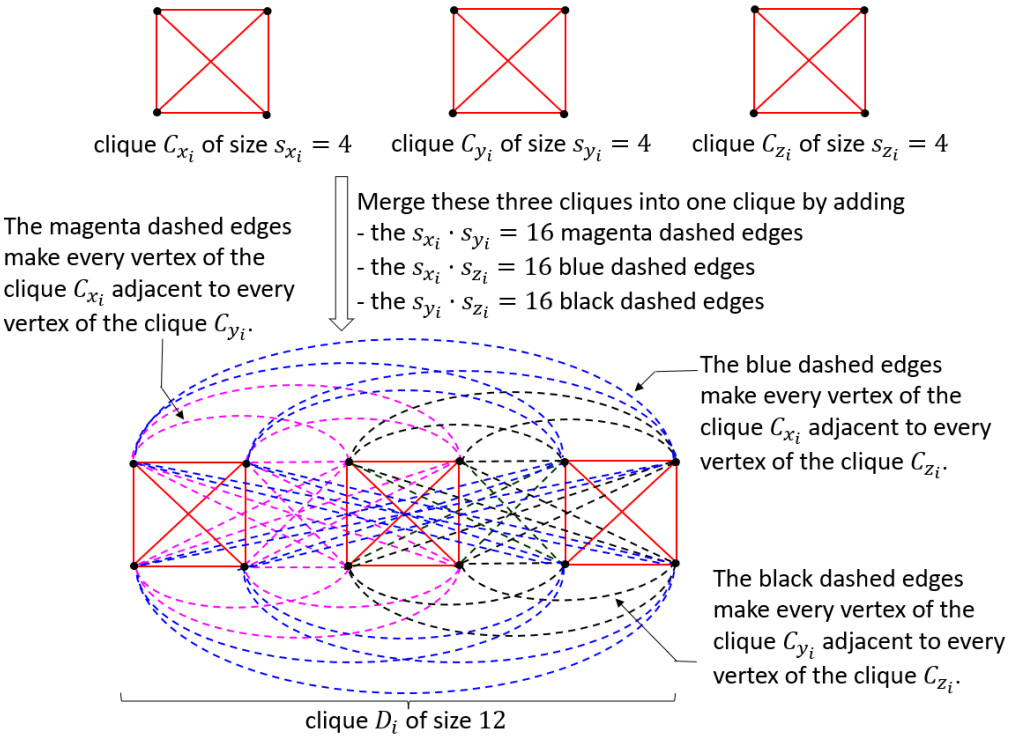


Figure 1 An illustration of the merger of the three cliques $C_{x_i}, C_{y_i}, C_{z_i}$, when $s_{x_i} = s_{y_i} = s_{z_i} = 4$, in Theorem 10.

Note that the number of edges so added to G is

$$\sum_{i=1}^{n} \left(s_{x_i} \cdot s_{y_i} + s_{x_i} \cdot s_{z_i} + s_{y_i} \cdot s_{z_i}\right) < \sum_{i=1}^{n} \left(3 \cdot \frac{b}{2} \cdot \frac{b}{2}\right) < nb^2$$

For each $1 \leqslant i \leqslant n$, the size of the clique D_i is $s_{x_i} + s_{y_i} + s_{z_i} = b$. The resulting graph, say H, is the disjoint union of $n + M$ cliques, each of size b, namely $D_1, \ldots, D_n$ and the M dummy cliques. The adjacency matrix of H has two distinct eigenvalues, i.e., -1 and $b - 1$. Thus, (G, nb^2) is a **YES** instance of 2-Eigenvalue Edge Addition.

($\Leftarrow$) Suppose that (G, nb^2) is a **YES** instance of 2-Eigenvalue Edge Addition. That is, there exists $S \subseteq \binom{V(G)}{2} \setminus E(G)$ of size $\leqslant nb^2$ such that adding the edges of S to G results in a graph, say H, whose adjacency matrix has at most two distinct eigenvalues. Using Lemma 1, the graph H is a disjoint union of equal-sized cliques. Observe that each clique of H is formed by merging some of the $3n + M$ cliques of G, namely $C_1, \ldots, C_{3n}$ and the M b-sized dummy cliques.

First, let us show that no dummy clique participates in a merger. That is, in H, each of the M b-sized dummy cliques of G remains as it is. For the sake of contradiction, assume that there exists a dummy clique that merges with some other clique(s) of G to form a bigger (i.e., of size $> b$) clique of H. Then, as all cliques of H have the same size, none of the other

$M - 1$ b-sized dummy cliques of G can remain as it is. Now, as each of the M b-sized dummy cliques participates in some merger, it is incident to $\geqslant b$ edges of S. Also, every edge of S is incident to at most two dummy cliques. Therefore, we get $|S| \geqslant \frac{Mb}{2} > nb^2$, a contradiction.

Let $D_1, \ldots, D_t$ denote the equal-sized cliques of H other than the M dummy cliques. Note that their common size is the same as that of a dummy clique, i.e., b. Consider any $1 \leqslant i \leqslant t$. The clique D_i is formed by merging some (say p_i) of the $3n$ cliques $C_1, \ldots, C_{3n}$. Each of these p_i cliques has size $> \frac{b}{4}$ and $< \frac{b}{2}$. Also, their sizes add up to the size of the clique D_i, i.e., b. Therefore, we have $p_i \cdot \frac{b}{4} < b < p_i \cdot \frac{b}{2}$. So, we get $p_i = 3$. Hence, each of the t cliques $D_1, \ldots, D_t$ is obtained by merging three of the $3n$ cliques $C_1, \ldots, C_{3n}$. We have $t = n$.

Consider any $1 \leqslant i \leqslant n$. Let $C_{x_i}, C_{y_i}, C_{z_i}$ denote the three cliques amongst $C_1, \ldots, C_{3n}$ whose merger forms the clique D_i. Let T_i denote the triplet that consists of $s_{x_i}, s_{y_i}, s_{z_i}$. As the sizes of the cliques $C_{x_i}, C_{y_i}, C_{z_i}$ add up to the size of the clique D_i, we have $s_{x_i} + s_{y_i} + s_{z_i} = b$. That is, the elements of the triplet T_i add up to b. Thus, as $T = T_1 \uplus \ldots \uplus T_n$, it follows that (T, b) is a **YES** instance of 3-Partition.

In the reduction above, instead of adding a clique on s_i vertices, we could instead add a cycle (resp. path) on s_i vertices, and adjust the budget to account for the missing edges, thereby showing NP-completeness on 2-regular graphs (resp. forests) as well. ◀

Our next result gives a quadratic kernel for 2-Eigenvalue Edge Addition. Let (G, k) be an instance of 2-EEA. We only describe the main intuition of the kernel informally and defer a detailed argument to a full version of this paper. Since we are only allowed to add edges, we "might as well" complete all the connected components of G to cliques and adjust the budget accordingly. Thus, without loss of generality, G is already a cluster graph. Some trivial cases are easily handled, such as: when we cannot afford to complete the original components of G to cliques, or when we have no budget but cliques of different sizes, or when all cliques are already of the same size.

Now, we are left with a situation where we have a non-trivial budget and cliques of at least two distinct sizes. Let the largest sized clique have q vertices, and suppose we have t cliques of size p in G, denoted by $C_1, \ldots, C_t$, where $p < q$. Note that each of these cliques is merged into a larger clique after edges from any valid solution are added to G. In particular, if S is a valid solution, at least $\frac{t \cdot p}{2}$ edges of S are incident to vertices of $C_1 \cup \ldots \cup C_t$. Therefore, if $tp/2 > k$, we can say **NO**. This bounds the sizes of cliques with fewer than q vertices.

For the largest-sized cliques, note that if we have "too many" of them, then none of them are merged into a larger clique after edges from any valid solution are added to G. In particular, it can be shown that if there are s cliques of size q, then if $sq > 2k$, then these cliques are untouched by any valid edge addition set of size at most k. This allows us to throw away most of them, preserving just enough to remember that the cliques must indeed remain untouched in any valid solution. This bounds the number of vertices among the largest sized clique.

Combining these arguments, the overall bound on the total number of vertices in the reduced instance turns out to be quadratic in k. We defer the a detailed proof to a full version of this paper.

▶ **Theorem 11.** *2-Eigenvalue Edge Addition admits a kernel with $O(k^2)$ vertices.*

Proof. Consider an instance, say (G, k), of 2-Eigenvalue Edge Addition. Owing to Lemma 1, our goal is to decide if we can add $\leqslant k$ edges to G to get a disjoint union of equal-sized cliques. Let us apply the following reduction rules (in the specified order):

Reduction rule 1: Suppose that there's a component, say C, of G, that is not a clique. Then, add the missing $\binom{|V(C)|}{2} - |E(C)|$ edges to turn C into a clique, and reduce the parameter k by $\binom{|V(C)|}{2} - |E(C)|$.

After exhaustively applying Reduction rule 1, G is a disjoint union of cliques; say, it consists of n_1 cliques of size x_1, n_2 cliques of size x_2, ..., n_t cliques of size x_t, where $x_1 < x_2 < \ldots\ldots < x_t$.

Reduction rule 2:

- If $k < 0$, then return **NO**.
- If $k \geqslant 0$ and $t = 1$, then return **YES**.
- If $k = 0$ and $t \geqslant 2$, then return **NO**.

After applying Reduction rule 2, we have $k \geqslant 1$ and $t \geqslant 2$.

Reduction rule 3: If there exists an $1 \leqslant i \leqslant t-1$ such that $n_i \cdot x_i > 2k$, then return **NO**.

Safeness of Reduction rule 3: Suppose that (G, k) is a **YES** instance. Then, there exists $S \subseteq \binom{V(G)}{2} \setminus E(G)$ of size $\leqslant k$ such that adding the edges of S to G results in a disjoint union of equal-sized (say, of size x) cliques. Observe that each of these x-sized cliques is obtained by merging some cliques of G. Note that x is at least the size of a largest clique in G. That is, we have $x \geqslant x_t$. Also, each of the smaller cliques of G, i.e., those of sizes $x_1, \ldots, x_{t-1}$, must participate in some merger.

Now, consider any $1 \leqslant i \leqslant t-1$. Each of the n_i cliques of size x_i is incident to $\geqslant x_i$ edges of S, for it must participate in some merger. Also, any edge of S is incident to at most two of these n_i cliques. Therefore, $|S| \geqslant \frac{n_i \cdot x_i}{2}$. So, as $|S| \leqslant k$, we get $n_i \cdot x_i \leqslant 2k$. Thus, Reduction rule 3 is safe.

After applying Reduction rule 3, we have $n_i \cdot x_i \leqslant 2k$ for all $1 \leqslant i \leqslant t-1$. Also, as $x_1, \ldots, x_{t-1}$ are $t-1$ distinct integers in the interval $[1, 2k]$, we get $t - 1 \leqslant 2k$.

Reduction rule 4: Suppose that $n_t \cdot x_t > 2k$. Then, remove all but $\frac{2k+1}{x_t}$ cliques of size x_t from G.

Safeness of Reduction rule 4: If $n_t \cdot x_t > 2k$, then in any solution, none of the n_t cliques of size x_t participate in a merger. That is, each of them remains as is after the edge additions, and each merger (involving the remaining cliques, i.e., those of sizes $x_1, \ldots, x_{t-1}$) results in an x_t-sized clique. This is because if any clique of size x_t gets to participate in a merger, then each of the remaining $n_t - 1$ cliques of size x_t must also participate in some merger (because all cliques have the same size after the edge additions), thereby needing $\geqslant \frac{n_t \cdot x_t}{2} > k$ edge additions.

Also, we have $n_t \cdot x_t > 2k$ before, as well as after, applying Reduction rule 4. Therefore, it follows that any solution before applying RR4 remains a solution after applying Reduction rule 4, and vice versa. Thus, Reduction rule 4 is safe.

If Reduction rule 4 wasn't invoked, then $n_t \cdot x_t \leqslant 2k$; otherwise, after applying Reduction rule 4, we get $n_t \cdot x_t = 2k + 1$.

Finally, the number of vertices in G is at most

$$n_1 \cdot x_1 + \ldots\ldots + n_{t-1} \cdot x_{t-1} + n_t \cdot x_t \leqslant (t-1) \cdot 2k + (2k+1) \leqslant 4k^2 + 2k + 1.$$

This concludes the proof of Theorem 11. ◀

Next, we show that r-EEA is NP-complete for every fixed $r \geqslant 3$.

▶ **Theorem 12** (⋆). *Let $r \geqslant 3$ be an integer. Then, r-Eigenvalue Edge Addition is NP-complete.*

5 Reducing eigenvalues by deleting edges

In this section, we consider the r-Eigenvalue Edge Deletion problem. We defer the NP-completeness of 2-EED to the proof of Theorem 16, where the hardness is implicit. In this section, we present an $\mathcal{O}^*(2^k)$-time FPT algorithm for 2-EED and show that it can be solved in polynomial time on triangle-free graphs. Finally, we prove that r-EED is NP-complete for any fixed $r \geqslant 3$.

The FPT algorithm is similar in spirit to the one we use in the proof of Theorem 6: we branch on induced paths of length three, except we now have a choice of two edges instead of three vertices. In particular, if P is an induced path on $\{a, b, c\}$ with edges $\{a, b\}$ and $\{b, c\}$, we recursively solve the instances $(G \setminus \{a, b\}, k-1)$ and $(G \setminus \{b, c\}, k-1)$.

At the leaves of successful execution paths of this branching algorithm, as before, we have cluster graphs where the cliques are not necessarily of the same size, and a residual budget. Let (H, k') denote such an instance, where H is a subgraph of G consisting of t cliques of sizes $s_1, \ldots, s_t$, and $k' \leqslant k$ is the residual budget. Note that if S is such that $G \setminus S$ is a collection of x-sized cliques for some x, then x must divide each s_i. We show that for an optimal choice of S, x is the GCD of the s_i's. Based on this, it is straightforward to check if the residual budget is sufficient or not.

▶ **Theorem 13.** *2-Eigenvalue Edge Deletion admits an algorithm with running time* $\mathcal{O}^*(2^k)$.

Proof. Let us describe a recursive branching algorithm. Consider an instance, say (G, k), of 2-Eigenvalue Edge Deletion. Owing to Lemma 1, our goal is to decide whether we can delete at most k edges from G to get a disjoint union of equal-sized cliques. First, we check if G has an induced path on three vertices. This takes polynomial time.

Case 1: G has no induced path on three vertices. The graph G is a disjoint union of cliques, say $C_1, \ldots, C_t$, of sizes $s_1, \ldots, s_t$ respectively. Observe that deleting the edges of any solution breaks each of these t cliques into equal-sized cliques (say, of size x). That is, for every $1 \leqslant i \leqslant t$, it breaks the clique C_i into $\frac{s_i}{x}$ cliques, each of size x. As each of these $\frac{s_i}{x}$ cliques has $\binom{x}{2}$ edges, the number of edges deleted from the clique C_i is

$$\binom{s_i}{2} - \frac{s_i}{x}\binom{x}{2} = \frac{s_i(s_i - x)}{2}$$

So, larger x corresponds to smaller solutions, i.e., fewer edge deletions. Also, x must divide each of $s_1, \ldots, s_t$. Therefore, for any minimum-sized solution, we have $x = \gcd(s_1, \ldots, s_t)$, and its size is

$$\sum_{i=1}^{t} \frac{s_i(s_i - \gcd(s_1, \ldots, s_t))}{2}$$

If this size is at most k, we return **YES**; otherwise, we return **NO**. This takes polynomial time. See Figure 2 for an example.

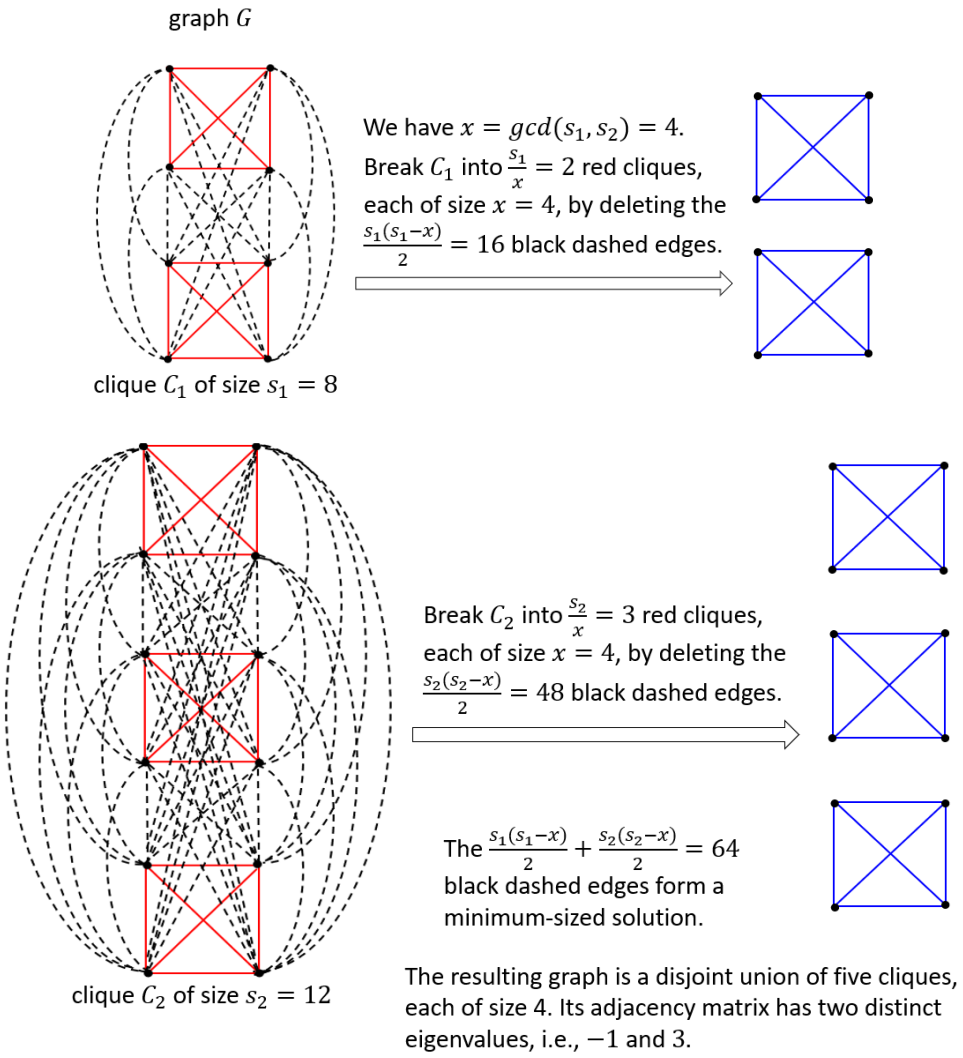


Figure 2 An example illustrating the breaking of cliques in Theorem 13.

Case 2: G has an induced path on three vertices, say $a - b - c$. Note that any solution must pick at least one of its two edges, i.e., $\{a, b\}$ and $\{b, c\}$. So, if $k = 0$, we return NO; otherwise, we guess an edge that is picked into solution. That is, we branch as follows: In the first (resp. second) branch, we include the edge $\{a, b\}$ (resp. $\{b, c\}$) into solution, remove it from G, and reduce the parameter k by 1. It takes polynomial time to create the sub-problems $(G - \{a, b\}, k - 1)$ and $(G - \{b, c\}, k - 1)$. Next, we run our algorithm on these two instances. If at least one of these two recursive calls returns **YES**, so do we; otherwise, we return **NO**.

The depth of our search tree is at most k. Also, each of its internal nodes has two children. Therefore, it has at most $\mathcal{O}(2^k)$ nodes. Thus, as we spend polynomial time at each node, the overall running time is at most $\mathcal{O}^\star(2^k)$. This concludes the proof of Theorem 13. ◀

Our next claim takes advantage of the fact that the sizes of the cliques after the removal of any solution is at most two when the input graph is triangle-free and we are only allowed to delete edges. Therefore, the value of the optimal solution is $|E(G)| - |V(G)|/2$ if G has a perfect matching and $|E(G)|$ otherwise. The result follows from the fact that the existence of a perfect matching can be determined in polynomial time [6].

▶ **Proposition 14.** *2-Eigenvalue Edge Deletion is polynomial time solvable on triangle-free graphs.*

Now, we show that r-EED is NP-complete by reducing it from Partition into Triangles on graphs of clique number 3 which is known to be NP-complete [4]. The input for Partition into Triangles is a graph G, and the goal is to decide whether $V(G)$ can be partitioned into $\frac{|V(G)|}{3}$ triplets such that every triplet induces a triangle in G.

▶ **Theorem 15.** *Let $r \geqslant 3$ be an integer. Then, r-Eigenvalue Edge Deletion is NP-complete.*

Proof. Let us describe a polynomial-time many-one reduction from Partition into Triangles on graphs of clique number 3 to r-Eigenvalue Edge Deletion. Consider an instance, say G, of Partition into Triangles, where G is a graph, say on n vertices and m edges, with clique number 3. Let us construct a graph, say H, from G, as follows: First, we add G as it is. Next, for each $3 \leqslant i \leqslant r+1$, we introduce $M := m - n + 1$ cliques, each of size i; let us refer to these cliques as *dummy cliques*. That is, the graph H is the disjoint union of the graph G, M dummy cliques of size 3, ..., M dummy cliques of size $r+1$. We set the budget to be $m - n$. Let us show that G has $\frac{n}{3}$ pairwise vertex disjoint triangles if and only if $(H, m-n)$ is a YES instance of r-Eigenvalue Edge Deletion.

($\Rightarrow$) Suppose that G has $\frac{n}{3}$ pairwise vertex disjoint triangles, say $T_1, \ldots, T_{n/3}$. Let S denote the set that consists of those $m-n$ edges of G that do not belong to any of these $\frac{n}{3}$ triangles. Note that the graph $H \setminus S$ is the disjoint union of

- $\frac{n}{3} + M$ triangles, namely $T_1, \ldots, T_{n/3}$ and the M dummy cliques of size 3. They contribute two distinct eigenvalues, i.e., -1 and 2.
- M dummy cliques of size 4. They contribute two distinct eigenvalues, i.e., -1 and 3.

⋮

- M dummy cliques of size $r+1$. They contribute two distinct eigenvalues, i.e., -1 and r.

So, the adjacency matrix of the graph $H \setminus S$ has r distinct eigenvalues, namely $-1, 2, 3, \ldots\ldots, r$. Thus, $(H, m-n)$ is a **YES** instance of r-Eigenvalue Edge Deletion.

($\Leftarrow$) Suppose that $(H, m-n)$ is a **YES** instance of r-Eigenvalue Edge Deletion. That is, there exists $S \subseteq E(H)$ of size $\leqslant m-n$ such that the adjacency matrix of the graph obtained by deleting the edges of S from H has $\leqslant r$ distinct eigenvalues.

Consider any $3 \leqslant i \leqslant r+1$. Note that the number of i-sized dummy cliques, i.e., M, is $> m - n \geqslant |S|$. So, there's at least one i-sized dummy clique, say C_i, such that none of its edges is deleted. That is, no edge of C_i belongs to S and thus, it appears as a component of the graph $H \setminus S$, thereby contributing two distinct eigenvalues, namely -1 and $i-1$. Thus, it follows that the adjacency matrix of the graph $H \setminus S$ must have $-1, 2, 3, \ldots, r$ as its r distinct eigenvalues.

Now, using Lemma 4, it is clear that the graph $H \setminus S$ must be a disjoint union of some cliques, whose sizes are $3, 4, \ldots, r+1$. So, as G has clique number 3, after removing those edges of G that belong to S, we're left with $\frac{n}{3}$ pairwise vertex-disjoint triangles of G, as desired. This concludes the proof. ◀

6 Reducing eigenvalues by editing edges

In this section, we show that 2-Eigenvalue Edge Editing is NP-complete. We give a reduction from Partition into Triangles.

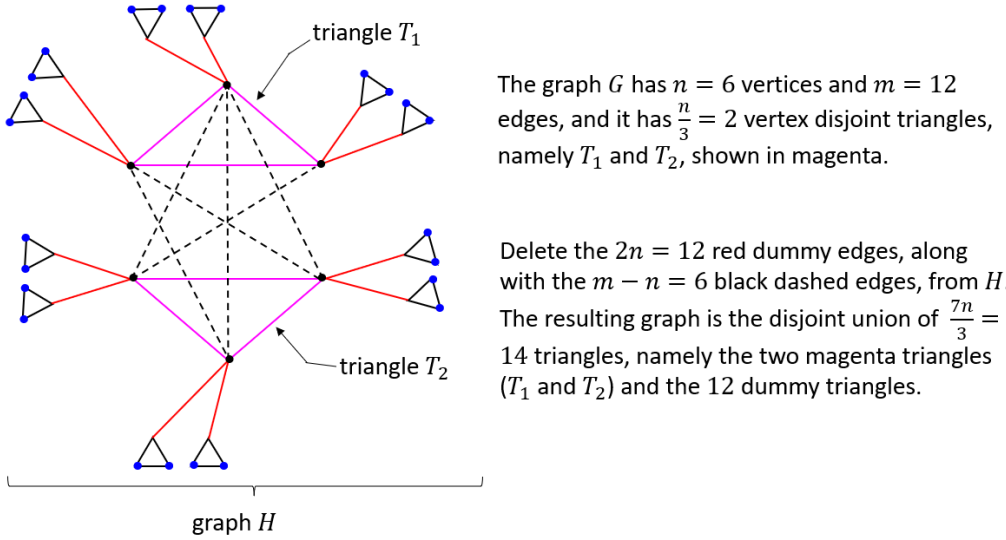
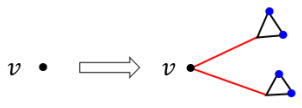


Figure 3 An example illustrating the construction in Theorem 16.

▶ **Theorem 16.** *2-Eigenvalue Edge Editing is NP-complete.*

Proof. Let us describe a polynomial-time many-one reduction from Partition into Triangles to 2-Eigenvalue Edge Editing. Consider an instance, say G, of Partition into Triangles, where G is a graph on n vertices and m edges. Let us construct a graph H based on G as follows: for every vertex $v \in V(G)$, attach two triangles to v, as shown below.



Let's refer to these two triangles as *dummy triangles*, and the two red edges that join the vertex v to these triangles as *dummy edges*. Also, let's refer to the four blue vertices as *saviour vertices*.

See Figure 3 for an illustration.

Note that $|V(H)| = 7n$ and $|E(H)| = m + 8n$. Let us show that G has $\frac{n}{3}$ pairwise vertex disjoint triangles if and only if $(H, m + n)$ is a **YES** instance of 2-Eigenvalue Edge Editing.

($\Rightarrow$) Suppose that G has $\frac{n}{3}$ pairwise vertex disjoint triangles, say $T_1, \ldots, T_{n/3}$. Let $S \subseteq E(H)$ denote the set that consists of the $2n$ dummy edges, along with those $m - n$ edges of G that do not belong to any of these $\frac{n}{3}$ triangles. Note that the graph $H \setminus S$ is the disjoint union of $\frac{7n}{3}$ triangles, namely $T_1, \ldots, T_{n/3}$ and the $2n$ dummy triangles. Its adjacency matrix has two distinct eigenvalues, i.e., -1 and 2. Thus, $(H, m + n)$ is a **YES** instance of 2-Eigenvalue Edge Editing.

($\Leftarrow$): Suppose that $(H, m+n)$ is a **YES** instance of 2-Eigenvalue Edge Editing. That is, there exist $D \subseteq E(H)$ and $A \subseteq \binom{V(H)}{2} \setminus E(H)$ such that: i) $|A| + |D| \leqslant m + n$, and ii) deleting the edges of D from H, and adding the edges of A to H, results in a graph, say H', whose adjacency matrix has at most two distinct eigenvalues. Using Lemma 1, the graph H' is a disjoint union of equal-sized cliques (say, of size x). As each of these $\frac{|V(H)|}{x}$ cliques has $\binom{x}{2}$ edges, the number of edges in H' is

$$\frac{|V(H)|}{x} \cdot \binom{x}{2} = \frac{7n(x-1)}{2}$$

Also, we have $|E(H)| + |A| - |D| = |E(H')|$. Therefore,

$$(m + 8n) + |A| - |D| = \frac{7n(x-1)}{2} \tag{1}$$

Adding (1) to the inequality $|A| + |D| \leqslant m + n$, we get

$$|A| \leqslant \frac{7n(x-3)}{4} \tag{2}$$

Note that each saviour vertex has degrees 2 and $x-1$ in H and H' respectively. So, each of the $4n$ saviour vertices is incident to $\geqslant x-3$ added edges (i.e., edges of A). Also, any edge of A is incident to at most two saviour vertices. Therefore,

$$|A| \geqslant \frac{4n(x-3)}{2} \tag{3}$$

Using (2) and (3), we get $x = 3$ and $|A| = 0$. Thus, the graph H' is a disjoint union of $\frac{7n}{3}$ triangles, obtained from H by only edge deletions: in other words, no edge additions are involved. This implies that we have $\frac{7n}{3}$ pairwise vertex disjoint triangles, say $T_1, \ldots, T_{7n/3}$, of the $7n$-vertex graph H. Note that the vertices of any dummy triangle belong to a unique triangle (i.e., the dummy triangle itself) in H. So, amongst $T_1, \ldots, T_{7n/3}$, we must have the $2n$ dummy triangles. Now, it is clear that the remaining $\frac{7n}{3} - 2n = \frac{n}{3}$ triangles form a collection of pairwise vertex disjoint triangles in G, as desired. ◀

7 Concluding Remarks

We considered the problem of modifying a graph optimally to reduce the number of distinct eigenvalues in the spectrum of its adjacency matrix. These problems turned out to be closely related to, but different from, modifications that aim to reduce the rank of the adjacency matrix and the diameter of the graph.

The complexity of r-EEE for fixed $r \geqslant 3$ remains open. The parameterized complexity of 2-EEE in the standard parameter is open, and the question of finding polynomial kernels for 2-EVD and 2-EED remains open as well. Studying these problems from the perspective of structural parameters or on directed graphs are interesting directions for future work.

References

1. Helmut Alt, Norbert Blum, Kurt Mehlhorn, and Markus Paul. Computing a maximum cardinality matching in a bipartite graph in time o (n1. 5mlog n). *Information Processing Letters*, 37(4):237–240, 1991.
2. Andries E Brouwer and Willem H Haemers. *Spectra of graphs*. Springer Science & Business Media, 2011.
3. Dibyayan Chakraborty, L Sunil Chandran, Sajith Padinhatteeri, and Raji R Pillai. Algorithms and complexity of s-club cluster vertex deletion. In *Combinatorial Algorithms: 32nd International Workshop, IWOCA 2021*, pages 152–164. Springer, 2021.
4. Ante Ćustić, Bettina Klinz, and Gerhard J Woeginger. Geometric versions of the three-dimensional assignment problem under general norms. *Discrete Optimization*, 18:38–55, 2015.
5. Michael Doob. On characterizing certain graphs with four eigenvalues by their spectra. *Linear Algebra and its applications*, 3(4):461–482, 1970.
6. Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.
7. Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

8 Felix Goldberg, Steve Kirkland, Anu Varghese, and Ambat Vijayakumar. On split graphs with four distinct eigenvalues. *Discrete Applied Mathematics*, 277:163–171, 2020.

9 S.M. Meesum, Pranabendu Misra, and Saket Saurabh. Reducing rank of the adjacency matrix by graph modification. *Theoretical Computer Science*, 654:70–79, 2016.

10 Syed M. Meesum and Saket Saurabh. Rank reduction of oriented graphs by vertex and edge deletions. *Algorithmica*, 80(10):2757–2776, 2018.

11 Assaf Natanzon. *Complexity and approximation of some graph modification problems*. University of Tel-Aviv, 1999.

12 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

13 Michele Zito. Linear time maximum induced matching algorithm for trees. *Nord. J. Comput.*, 7(1):58, 2000.

Connected Vertex Cover on AT-Free Graphs

Joydeep Mukherjee
Ramakrishna Mission Vivekananda Educational and Research Institute, Belur, India

Tamojit Saha
Ramakrishna Mission Vivekananda Educational and Research Institute, Belur, India
Institute of Advancing Intelligence, TCG CREST, Kolkata, India

Abstract

Asteroidal Triple (AT) in a graph is an independent set of three vertices such that every pair of them has a path between them avoiding the neighbourhood of the third. A graph is called AT-free if it does not contain any asteroidal triple. A connected vertex cover of a graph is a subset of its vertices which contains at least one endpoint of each edge and induces a connected subgraph. Settling the complexity of computing a minimum connected vertex cover in an AT-free graph was mentioned as an open problem in Escoffier et al. [6]. In this paper we answer the question by presenting an exact polynomial time algorithm for computing a minimum connected vertex cover problem on AT-free graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graph Algorithm, AT-free graphs, Connected Vertex Cover, Optimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.54

1 Introduction

An *Asteroidal Triple (AT)* of a graph $G = (V, E)$ is a set of three vertices of $V(G)$ such that these three vertices are mutually nonadjacent and for any two vertices of this set there exists a path between these two vertices which avoids the neighborhood of the third vertex. A graph is called *asteroidal triple free (AT-free)* if it does not contain any asteroidal triple.

We assume, in the rest of the paper, the graph G is undirected, unweighted and simple graph. A subset of $V(G)$ is a vertex cover of G if every edge of G has an endpoint in that subset. The minimum vertex cover problem is to find a vertex cover of minimum cardinality. A vertex cover which also induces a connected subgraph of G is called a connected vertex cover. The minimum connected vertex cover problem is to find a vertex cover of minimum cardinality such that the vertices of the vertex cover induces a connected subgraph. In the rest of the paper we denote the minimum vertex cover problem by MVC and the minimum connected vertex cover problem by MCVC.

In this paper we present a polynomial time algorithm for MCVC on connected AT-free graphs. More precisely we provide an $O(n^4)$ algorithm for minimum connected vertex cover in AT-free graphs. In [3] Broersma et al. presented a polynomial time algorithm for maximum independent set problem. Our work is inspired by the technique developed in that paper. In the following we define the problem more formally.

Connected Vertex cover On AT-free graphs
Instance: A connected AT-free graph $G = (V, E)$, $|V| = n$, $|E| = m$.
Output: A set $S^* \subseteq V(G)$ of minimum cardinality such that $G[S^*]$ is connected and S^* contains at least one end point of every edge in G, i.e. S^* is a vertex cover.

The MCVC problem is studied in several graph classes, and there exist various algorithms for this problem in the fields of approximation algorithm, fixed parameter algorithm, and polynomial time exact algorithm. In the following we discuss some of the known results for

this problem. This problem was first introduced by Garey and Johnson [8]. This problem is known to be NP-hard in planar bipartite graphs of maximum degree 4 [7], in planar bi-connected graphs of maximum degree 4 [17], in $H-$free graphs if H contains a cycle or a claw [16], and in 3-connected graphs [20]. It is APX-complete in bipartite graphs of maximum degree 4, even if each vertex of one partite set has a degree at most 3 [6].

MCVC is polynomial time solvable in many special graph classes like graphs of maximum degree 3 [19], $(sP_1 + P_5)-$free graphs [11]. Escoffier et al. [6] proved several results regarding the connected vertex cover problem in special graph classes. They showed this problem is polynomial time solvable in chordal graphs; in bipartite graphs if each vertex of one partite set has maximum degree 2 and the vertices of the other partite set have no restriction on the degree. They proved a PTAS for MCVC in planar graphs. In the same paper, they provided a $\frac{5}{3}-$approximation algorithm for MCVC on all those graphs for which MVC is solvable in polynomial time. On the complexity side they proved that MCVC is APX-hard in bipartite graphs. Note that results of this paper along with the polynomial time algorithm for independent set problem presented by Broersma et al. in [3], impliy a $\frac{5}{3}-$approximation algorithm for AT-free graphs. Escoffier et al. in the paper [6], posed the complexity of MCVC on AT-free graphs as an open problem.

We state known approximation algorithm and FPT algorithm results for MCVC. A 2-approximation algorithm for MCVC is known in general graphs [1, 18] but it is not possible to approximate MCVC within ratio $(10\sqrt{5} - 21)$ in general graphs unless $P = NP$ [7]. Several results for computing connected vertex cover are known in the field of fixed parameter algorithms. First result was an algorithm with running time $O(6^k)$ [10] which was later improved to $O(2.7060^k)$ [14], where k is the length of a minimum vertex cover in the given graph and also an algorithm with running time $O(2^t \cdot t^{(3t+2)}n)$ where t is the treewidth and n is the number of vertices in the given graph [15].

Asteroidal triple free graph class contains graph classes like permutation graphs, interval graphs, trapezoid graphs, and cocomparability graphs [5]. AT-free graphs have many desirable properties which make them amenable for designing polynomial time algorithms for many problems which are NP-complete in general graphs. Such problems include minimum feedback vertex set problem [13], maximum independent set [3], dominating set, total dominating set [12] and connected dominating set [2], induced disjoint path problem [9]. However, to the best of our knowledge, the complexity of computing connected vertex cover problem is unknown in AT-free graphs.

2 Preliminaries

Let $G = (V, E)$ be a simple unweighted graph. We denote the set of vertices by $V(G)$ and the set of edges by $E(G)$. A graph $H = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. We denote $|V|$ by n and $|E|$ by m. A subgraph $H = (V', E')$ of G is an induced subgraph if $V' \subseteq V$ and for $u, v \in V'$, $(u, v) \in E'$ if and only if $(u, v) \in E$. The induced subgraph on any subset $S \subseteq V$ is denoted by $G[S]$.

The neighbourhood of a vertex v, denoted by $N(v)$, is the set of all vertices that are adjacent to v. Closed neighbourhood of v is denoted by $N[v] = \{v\} \cup N(v)$. The neighbourhood of a set of vertices $\{v_1, v_2, \ldots, v_k\}$ is denoted by $N(v_1, v_2, \ldots, v_k) = \bigcup_{i=1}^{k} N(v_i)$ and the closed neighbourhood is denoted by $N[v_1, v_2, \ldots, v_k] = \bigcup_{i=1}^{k} N[v_i]$. Assume C is a connected component of G. The set $N_C(v)$ where $v \in V(C)$, denotes the set of neighbour of v that are in the component C.

A path is a graph, $Y = (V, E)$, such that $V = \{y_1, y_2, \ldots, y_k\}$ and $E = \{y_1y_2, y_2y_3, \ldots, y_{k-1}y_k\}$. We denote a path by the sequence of its vertices, that is $Y = y_1y_2 \ldots y_k$. Here y_1 and y_k are called endpoints of path Y. The number of ver-

tices present in Y is denoted by $|Y|$. We denote $y_iYy_j = y_iy_{i+1}\dots y_j$ where $1 \le i \le j \le k$. A path on k vertices is denoted by Y_k and the length of the path is denoted by the number of edges present on the path that is $k-1$. The distance between two vertices in a graph is the length of the shortest path between them. A cycle is a graph, $C = (V, E)$, such that $V(C) = \{c_1, c_2, \dots, c_l\}$ and $E(C) = \{c_1c_2, \dots, c_{l-1}c_l, c_lc_1\}$. The shortest distance between u and v is denoted by $dist_C(u, v)$ where $u, v \in V(C)$. The number of vertices present in the cycle C is denoted by $|C|$.

A *dominating set* D of G is a subset of vertices of G such that for every v outside D, $N(v) \cap D \neq \phi$. A *dominating pair* is a pair of vertices such that any path between them is a dominating set. There is a linear time algorithm to find a dominating pair [4] in AT-free graphs. We denote a shortest path between a dominating pair by DSP.

This paper is inspired by the technique developed by Broersma et al. in [3]. We use their method of graph decomposition and supplement it with our new observations for connected vertex cover in AT-free graphs to derive the results stated in this paper. Let $G(V, E)$ denote a connected AT-free graph. The components in the graph $G \setminus N[x]$, where $x \in V$, are denoted by $C_1^x, \dots, C_r^x$.

Let x and y be two nonadjacent vertices of the graph. We define an *interval* to be $I(x, y) \subseteq V(G)$ in the following :

$I(x, y) = \{s \in V(G) :$ there is a s, x-path which does not contain any neighbours of y and there is a s, y-path which does not contain any neighbour of $x\}$.

Assume a connected component C containing vertices x and y. We denote the interval $I(x, y)$ by $I_C(x, y)$ when we consider the induced subgraph on C instead of the whole graph G.

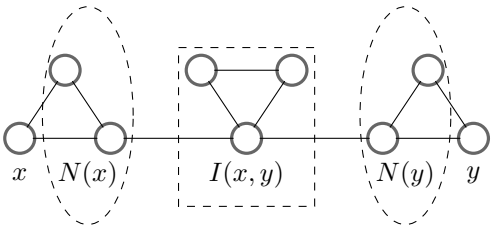


Figure 1 An example of interval in an AT-free graph.

Let $y \in V$ and the component of $G \setminus N[x]$ containing y is $C^x(y)$. The component in $G \setminus N[y]$ containing x is $C^y(x)$. The vertices in $C^x(y) \cap C^y(x)$ form a separator of x and y which is precisely the set $I(x, y)$. Note that $C^x(y) \cap C^y(x)$ may be empty.

In the following we state the necessary lemma from [3] for our purpose which provide us with some characterization for $I(x, y)$.

In the following lemma we consider a connected AT-free graph $G(V, E)$. We consider an interval $I(x, y)$ of G and assume that $s \in I(x, y)$. Lemma 1 is obtained using the fact that G is AT-free.

▶ **Lemma 1** (Broersma et al. [3]). *The vertices x and y are in different components of $G \setminus N[s]$ for each $s \in I(x, y)$.*

Thus every path between x and y either contains s or some neighbour of s. The next three lemma states a decomposition of an interval into disjoint intervals and disjoint components. Lemma 2 states that the intervals $I(x, s)$ and $I(s, y)$ have empty intersection. This implies that $P_{s,x} \cap P_{s,y} \subseteq N[s]$, where $P_{s,x}$ is an arbitrary $s, x-$path and $P_{s,y}$ is an arbitrary $s, y-$path in G.

▶ **Lemma 2** (Broersma et al. [3]). *The intervals $I(x, s)$ and $I(s, y)$ have no vertices in common, that is $I(x, s) \cap I(s, y) = \phi$.*

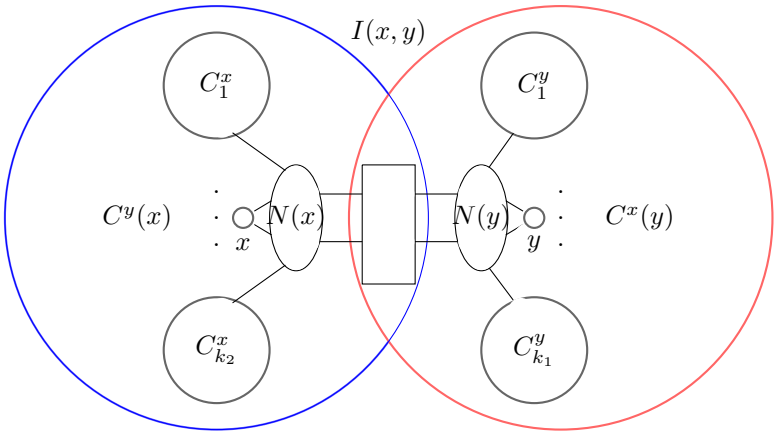


Figure 2 The component $C^y(x)$ which contains x in $G \setminus N[y]$.

Lemma 3 states a containment relation among the intervals. More precisely, if $s \in I(x,y)$ then $I(x,s) \subseteq I(x,y)$ and so does $I(s,y)$.

▶ **Lemma 3** (Broersma et al. [3]). *The intervals $I(x,s)$ and $I(s,y)$ are both contained in $I(x,y)$, that is $I(x,s) \subseteq I(x,y)$ and $I(s,y) \subseteq I(x,y)$, where $s \in I(x,y)$.*

Combining Lemma 2 and Lemma 3 we arrive at Lemma 4.

▶ **Lemma 4** (Broersma et al. [3]). *In the graph $G \setminus N[s]$ there are components $C_1^s, C_2^s, \ldots, C_t^s$ such that $I(x,y) \setminus N[s] = I(x,s) \cup I(s,y) \cup (\bigcup_{i=1}^{t} C_i^s)$.*

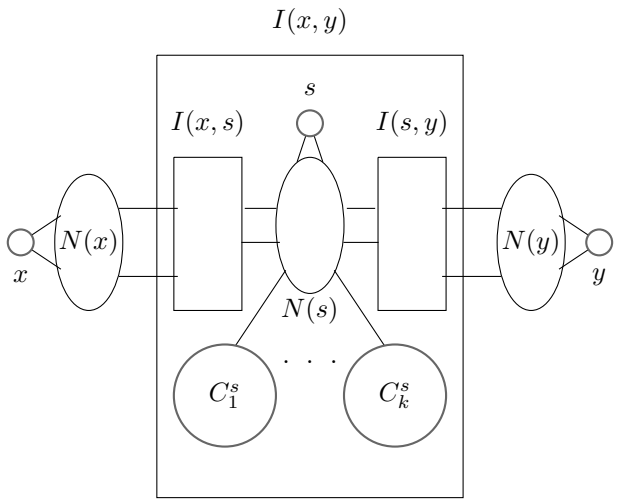


Figure 3 The interval decomposition.

Similarly the components of $G \setminus N[x]$ can also be decomposed. Consider such a component containing y, recall that $y \in C^x(y)$. The following lemma describes the structure of the graph induced on $C^x(y) \setminus N[y]$. In the following we denote the component of $G \setminus N[y]$ containing x by $C^y(x)$.

Consider the graph induced on $C^x(y) \setminus N[y]$ and let D be a connected component of that graph. Lemma 5 essentially states that any vertex of D reaches $N[x]$ using at least one vertex from $I(x,y)$.

▶ **Lemma 5** (Broersma et al. [3]). *Let D be a component of the graph $C^x(y) \setminus N[y]$. Then $N[D] \cap (N[x] \setminus N[y]) = \phi$ if and only if D is a component of $G \setminus N[y]$.*

3 Connected Vertex Cover

In the following sections we make some important observations related to the connectivity constraint of the vertex cover and then we formulate the dynamic programming recurrence relations.

3.1 Some structural observations

In this section and subsequent sections we assume $G(V, E)$ is a connected AT-free graph. Let α_c be an independent set with maximum cardinality, while ensuring that the subgraph $G[V \setminus \alpha_c]$ remains connected. The complement of α_c forms a connected vertex cover with the smallest possible size. Observe that α_c cannot include a cut vertex. This is because if a vertex v belongs to α_c, none of its neighbors can be in α_c. If v is a cut vertex, its neighbors would be divided into separate components, leading $G[V \setminus \alpha_c]$ to be disconnected. Hence we have the following observation.

▶ **Observation 6.** *Let α_c be an independent set with maximum cardinality, while ensuring that the subgraph $G[V \setminus \alpha_c]$ remains connected. The set α_c does not include any cut vertex of G.*

Let V' denote the set of all cut vertices in G. We define some notations that are necessary in the following set of lemma. Let x be a vertex of G which is not a cut vertex. Let $C_1^x, \ldots, C_r^x$ be the components of $G \setminus N[x]$. Let Z_i be those vertices of $N(x)$ which are reachable from C_i^x in the graph $G \setminus \{x\}$, that is without using the vertex x or vertices from any other components. In other words let C be the connected component in $G[N(x) \cup V(C_i^x)]$, then $Z_i = V(C) \setminus V(C_i^x)$. Note that $G[Z_i]$ may not be connected. Suppose S is a connected vertex cover of G and let $x \in V \setminus V'$ such that $x \notin S$. That is $N(x) \subseteq S$. Let $S_i = S \cap (Z_i \cup V(C_i^x))$. Note that S_i contains Z_i, since $Z_i \subseteq N(x)$.

▶ **Lemma 7.** *The graph induced on $S \cap (Z_i \cup V(C_i^x))$ is connected.*

Proof. Assume for the sake of contradiction, $G[S \cap (Z_i \cup V(C_i^x))]$ is not connected and $H_1, \ldots, H_k$ are the components of $G[S \cap (Z_i \cup V(C_i^x))]$. Each component H_j has $V(H_j) \cap Z_i \neq \phi$, because otherwise H_j is a component of $G \setminus N[x]$. Consider two components H_l, H_r. There is some vertex $v \in Z_i$ which is adjacent to some vertex of $v' \in V(H_l)$ and there is a vertex $u \in Z_i$ which is adjacent to some vertex $u' \in V(H_r)$. Note that, v is not adjacent to u and v' is not adjacent to v' since H_l and H_r are different components. Hence v', u', x forms an AT. The paths leading v', u', x to form an AT is as follows. The vertices v' and u' are in same component of $G \setminus N[x]$ but not adjacent, hence there is a u', v' path avoiding the neighbours of x. The path x, u, u' avoids the neighbours of v' and similarly the path x, v, v' avoids the neighbours of v'. ◀

Also note that the graph $S \cap (Z_i \cup V(C_i^x))$ is a vertex cover of the graph $G[Z_i \cup V(C_i^x)]$, since S is a vertex cover of G.

▶ **Lemma 8.** *Let S_i^* be a minimum connected vertex cover in $G[Z_i \cup V(C_i^x)]$ and let S^* be a minimum connected vertex cover in G. Then $|S_i^*| \leq |S^* \cap (Z_i \cup V(C_i^x))|$.*

Proof. Please find the proof in full version of the paper. ◀

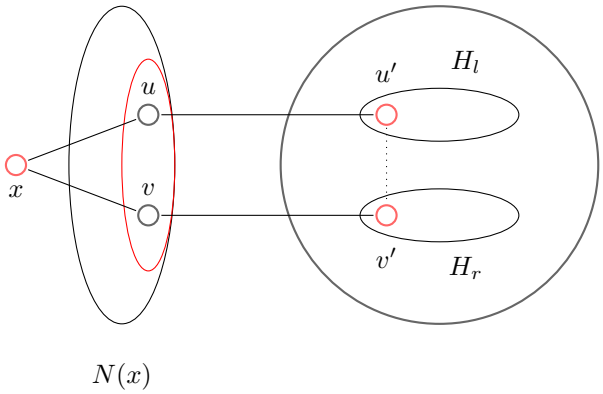


Figure 4 Illustrating proof of Lemma 7.

▶ **Lemma 9.** *Let S'_i denote a connected vertex cover in $G[Z_i \cup V(C_i^x)]$ containing Z_i. The graph induced on the set $N(x) \cup \left(\bigcup_{i=1}^{r} S'_i \right)$ is connected.*

Proof. Please find the proof in full version of the paper. ◀

3.2 The Dynamic Programming Formulation

The definition of intervals implies that, the set $I(x, y)$ is unique for each pair of non adjacent vertices x and y. The above property implies that the number of intervals is bounded by a polynomial in $|V(G)|$. We shall use these intervals to decompose the AT-free graph into smaller disjoint graphs. In the following sections, using this broad idea, we frame the recurrences to find an independent set of maximum size such that its complement is connected. We begin by a graph modification to incorporate the recurrence relation in terms of the intervals.

3.2.1 Graph modification

We begin by constructing a modified graph. A result by Corneil et al. [5], ensures that there exists a dominating pair in every AT-free graph which is *pokable*, that is we can append pendant vertices to both of the vertices of the pair maintaining the AT-free property. The following theorem by Corneil et al. [5] states that the process of composing two AT-free graphs.

▶ **Theorem 10** (The Composition Theorem; Corneil et al. [5])**.** *Given two AT-free graphs G_1 and G_2, and pokable dominating pairs (x_1, y_1) and (x_2, y_2) in G_1 and G_2, respectively, let G' be the graph constructed from G_1 and G_2 by identifying vertices x_1 and x_2. Then, G' is an AT-free graph.*

Let $p_1, \ldots, p_k$ be a dominating path where p_1 and p_k is a pokable dominating pair in G. An edge is AT-free, hence we append an edge (u, v) to p_1, that is v is adjacent to p_1 and an edge (u', v') to p_k, that is v' is adjacent to p_k. We add edges between v and p_2, v' and p_{k-1}. We denote this graph by G'. More formally, $G'(V, E)$ where,

$$V(G') = V(G) \cup \{u, v, u', v'\}$$

$$E(G') = E(G) \cup \{(u, v), (u', v'), (v, p_1), (v, p_2), (v', p_k), (v', p_{k-1})\}$$

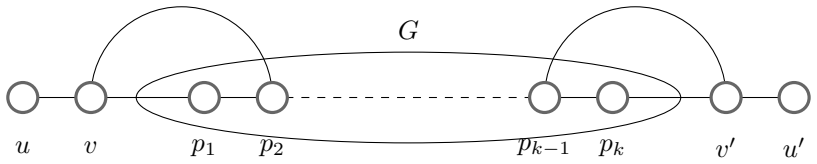


Figure 5 Illustrating graph modification.

We denote by α_c an independent set such that the remaining set of vertices induces a connected graph.

▶ **Lemma 11.** *The set α_c is a maximum independent set of G such that $G[V \setminus \alpha_c]$ is connected if and only if α_c is a maximum independent set of $G'[I(u,u')]$ such that $G'[V \setminus \alpha_c]$ is connected.*

Proof. The set $I(u,u')$ contains all the vertices that has a path to u avoiding the neighbourhood of u' and a path to u' avoiding the neighbourhood of u in the graph G'. From the construction of G', every vertex of $V(G)$ satisfies this property. Hence G and $I(u,u')$ are the same. The claim follows since G and $I(u,u')$ are the same. ◀

Now we define the recurrence relations for dynamic programming on the modified graph.

3.2.2 The dynamic programming

We decompose the graph in such a way that for any two non adjacent vertices x and y belonging to some connected component C has the property, $I_C(x,y) = I_G(x,y)$. More precisely we remove the closed neighbourhood of a vertex to achieve the smaller subgraphs. From Lemma 1, we can see that the invariant $I_C(x,y) = I_G(x,y) = I(x,y)$ is maintained while solving the subproblems. Broadly our approach is to compute minimum connected vertex cover in smaller connected components and take their disjoint union to obtain a minimum connected vertex cover of a larger component of which the smaller components are part. It is sufficient to calculate the minimum connected vertex cover for smaller components and combine them, which is ensured by Lemma 8 and Lemma 9. We begin by stating a recurrence for a component. In this recurrence we compute a maximum independent set of the component, such that the complement of this independent set (w.r.t the component) is connected. Note that we want the complement to be connected because of our observation in Lemma 7. The recurrence consists of decomposing a given component into interval and some connected components whose disjoint union is the given component as claimed in Lemma 2, Lemma 3 and Lemma 4.

Now we define required notations to state the recurrence formally. Suppose C is a connected component. Let x be a vertex in $V(C)$ which belongs to the independent set. Let the components in $C \setminus N[x]$ be denoted by $C_1^x, \ldots, C_k^x$ if $C \setminus N[x]$ has k connected components.

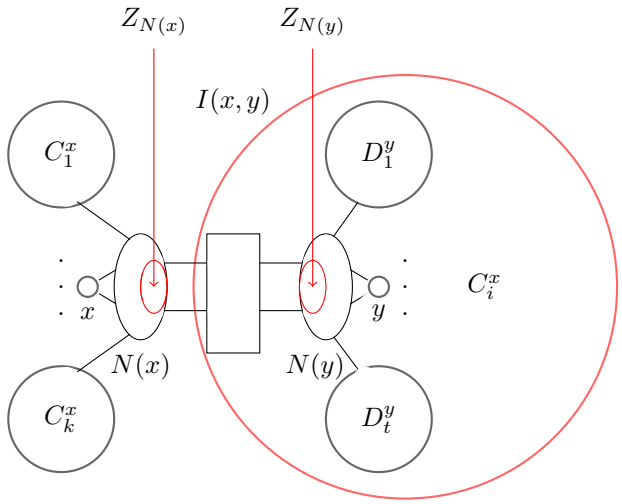


Figure 6 Illustrating Lemma 12.

In the following we define some notations which are necessary to state the recurrence.

- Consider a component C_i^x in $C \setminus N(x)$.
- Let $V'_{C_i^x}$ be the set of cut vertices of $C[V(C_i^x) \cup Z_{N(x)}]$. We choose $y \in V(C_i^x) \setminus V'_{C_i^x}$ as a candidate for the independent set since from Lemma 7 we know that $C[(V(C_i^x) \cup Z_{N(x)}) \setminus \{y\}]$ is connected.
- Let $I(x,y)$ be the interval for vertices x and y.
- Let $Z_{N(x)}$ be those vertices of $N(x)$ that are reachable from C_i^x in $C[N(x)]$ without using x or vertices from any other components.
- Let $Z_{N(y)}$ be those vertices of $N(y)$ that are reachable from $I(x,y)$ in $C[N(y)]$ without using y or vertices from any other components.
- Let $D_1^y, \ldots, D_t^y$ be the components of $C[C_i^x \setminus N[y]]$, and let H_j be those vertices of $N(y)$ that are reachable from $N(y) \cap N(V(D_j^y))$ in $C[N(y)]$ without using y or vertices from any other components.
- We define $\beta(C_i^x, Z_{N(x)})$ to be a maximum independent set in C_i^x such that $C[Z_{N(x)} \cup (V(C_i^x) \setminus \beta(C_i^x, Z_{N(x)}))]$ is connected.
- We define $\gamma(I(x,y), Z_{N(x)} \cup Z_{N(y)})$ to be a maximum independent set in $I(x,y)$ such that $G[Z_{N(x)} \cup Z_{N(y)} \cup (I(x,y) \setminus \gamma(I(x,y), Z_{N(x)} \cup Z_{N(y)}))]$ is connected.

▶ **Lemma 12.** *The recurrence for β is as follows.*

$$|\beta(C_i^x, Z_{N(x)})| = 1 + \max_{y \in C_i^x \setminus V'_{C_i^x}} \left(|\gamma(I(x,y), Z_{N(x)} \cup Z_{N(y)})| + \sum_{j=1}^{t} |\beta(D_j^y, H_j)| \right)$$

Proof. Please find the proof in the appendix. ◀

Note that if C is the whole graph then x is not a cut vertex of G.

Now we state recurrence for an interval. In this recurrence we compute a maximum independent set of the interval, such that the complement of this independent set (w.r.t the interval) is connected. Note that we want the complement to be connected because of our observation in Lemma 7. The recurrence consists of decomposing a given interval into disjoint sub intervals and some connected components whose disjoint union is the given interval as claimed in Lemma 2, Lemma 3 and Lemma 4.

▶ **Observation 13.** *The graph $G[I(x,y) \cup Z_{N(x)} \cup Z_{N(y)}]$ is connected.*

Proof. Please find the proof in full version of the paper. ◀

- Note that the definitions of x, y, $Z_{N(x)}$, $Z_{N(y)}$ and $\gamma(I(x,y), Z_{N(x)} \cup Z_{N(y)})$ remains same as earlier.
- Let $V'_{I(x,y)}$ be the set of cut vertices in $C[I(x,y) \cup Z_{N(x)} \cup Z_{N(y)}]$. We choose $s \in I(x,y) \setminus V'_{I(x,y)}$ as a candidate for the independent set in $I(x,y)$, since from Lemma 7, we know that $C[(I(x,y) \cup Z_{N(x)} \cup Z_{N(y)}) \setminus \{s\}]$ is connected.
- Let $A_{N(x)}$ be those vertices of $N(x)$ that are reachable from $I(x,s)$ in $C[N(x)]$ without using x and any vertex from other components.
- Let $A_{N(s)}$ be those vertices of $N(s)$ that are reachable from $I(x,s)$ in $C[N(s)]$ without using s and any vertex from other components.
- Let $B_{N(y)}$ be those vertices of $N(y)$ that are reachable from $I(y,s)$ in $C[N(y)]$ without using y and any vertex from other components.
- Let $B_{N(s)}$ be those vertices of $N(s)$ that are reachable from $I(s,y)$ in $C[N(s)]$ without using s and any vertex from other components.
- Let $Y^s_1, \ldots, Y^s_l$ are the components of $G[I(x,y) \setminus N[s]]$, and H_j are those vertices of $N(s)$ that are reachable from Y^s_j is $N(s)$ in $G[N(s)]$ without using the vertex s and vertices from other components.

We need the following lemma to prove the correctness of the recurrence for the intervals. Lemma 14 is similar to 7 and Lemma 15 is similar to Lemma 9.

▶ **Lemma 14.** *Let S be a connected vertex cover of G such that $x, y \notin S$. Then $S \cap (I(x,y) \cup Z_{N(x)} \cup Z_{N(y)})$ induces a connected subgraph.*

Proof. Please find the proof in full version of the paper. ◀

Let $S_{I(x,s)}$ denote vertices of a connected vertex cover in $G[Z_{N(x)} \cup I(x,s) \cup A_{N(s)}]$ such that $(Z_{N(x)} \cup A_{N(s)}) \subseteq S_{I(x,s)}$ and let $S_{I(s,y)}$ denote a connected vertex cover in $G[Z_{N(y)} \cup I(s,y) \cup B_{N(s)}]$ such that $(Z_{N(y)} \cup B_{N(s)}) \subseteq S_{I(s,y)}$. Let S_j denote a connected vertex cover in $G[H_j \cup V(Y^s_j)]$ containing H_j.

▶ **Lemma 15.** *The graph induced on the set $N(s) \cup \left(\bigcup_{i=1}^{r} S_i\right) \cup S_{I(x,s)} \cup S_{I(s,y)}$ is connected.*

Proof. Please find the proof in full version of the paper. ◀

Please see the Figure 7 for clarification of the following lemma. Note that $Z_{N(x)}$ and $A_{N(x)}$ are same and $Z_{N(y)}$ and $B_{N(y)}$ are same.

▶ **Lemma 16.** *The recurrence for γ is as follows. Let $Z = Z_{N(x)} \cup Z_{N(y)}$.*

$$|\gamma(I(x,y), Z)| =$$

$$1 + \max_{s \in I(x,y) \setminus V'_{I(x,y)}} \left(|\gamma(I(x,s), A_{N(x)} \cup A_{N(s)})| + |\gamma(I(s,y), B_{N(y)} \cup B_{N(s)})| + \sum_{j=1}^{s} |\beta(Y^s_j, H_j)| \right)$$

Proof. Please find the proof in the appendix. ◀

Consider the modified graph G'. Since (p_1, p_2) and (p_{k-1}, p_k) are edges of G (also of G'), the connected vertex cover must contain at least one endpoint from each of those edges. The $\gamma(I(u,u'), \{v,v'\})$ is a maximum independent set such that, $G'[\{v,v'\} \cup (I(u,u') \setminus \gamma(I(u,u'), \{v,v'\})]$ is connected. Since $I(u,u')$ is the graph G, $G'[V \setminus \gamma(I(u,u'), \{v,v'\})]$ is our desired solution from Lemma 11.

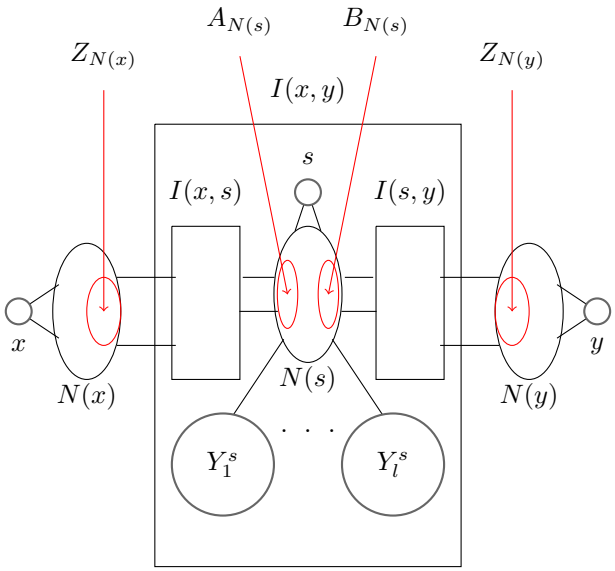


Figure 7 Illustrating Lemma 16.

3.3 Running time analysis

We employ dynamic programming technique to solve the above recurrences to obtain the minimum connected vertex cover. Let $\mathcal{C}$ be a set of components of G as defined below.

$$\mathcal{C} = \bigcup_{v \in V(G)} \{C : C \text{ is a component of } G \setminus N[v]\}$$

We prove in the following observation that the components for which we compute β are precisely the members of $\mathcal{C}$. The proof of the following observation is a repeated application of Lemma 5.

▶ **Observation 17.** *The non-interval components in the above recurrence relations are members of* $\mathcal{C}$.

Proof. Please find the proof in full version of the paper. ◀

Observation 17 ensures that the number of non-interval components is $|\mathcal{C}| = O(n^2)$.

Let $\mathcal{I}$ denote the set of all possible intervals. The collection $\mathcal{I}$ has cardinality at most n^2, that is $|\mathcal{I}| \leq n^2$, since $I(x, y)$ is unique for each pair of non adjacent vertex x and y. We arrange the list of all intervals and components in the non decreasing order of the number of vertices. We compute the recurrences for this two lists in the order they are arranged.

First we discuss the complexity to solve the recurrence for intervals. Consider a particular interval $I(x, y)$. We have to go through all the vertices in that $I(x, y)$ and there is at most $O(n)$ such vertices and for each vertex there can be at most $O(n)$ components. Note that these components are smaller than the component that $I(x, y)$ is part of. Hence the solution for each of the components are already stored in the dynamic programming table. Since there is at most $O(n^2)$ intervals and each can take $O(n^2)$ time it takes $O(n^4)$ to find the solutions.

The time complexity to solve the components is calculated similarly and it is also upper bounded by $O(n^4)$. Since all the other computation can be done in time $O(n^4)$ the complexity of our algorithm is $O(n^4)$.

4 Conclusion

In this paper, we present a polynomial time algorithm to compute a minimum connected vertex cover on AT-free graphs. Note that even though we have considered an unweighted graph, this algorithm can be modified in such a way that it also works for weighted AT-free graphs.

It will be interesting to explore the complexity of MCVC for those graph classes where MVC is solvable in polynomial time.

References

1 Esther M Arkin, Magnús M Halldórsson, and Rafael Hassin. Approximating the tree and tour covers of a graph. *Information Processing Letters*, 47(6):275–282, 1993.

2 Hari Balakrishnan, Anand Rajaraman, and C Pandu Rangan. Connected domination and steiner set on asteroidal triple-free graphs. In *Algorithms and Data Structures: Third Workshop, WADS'93 Montréal, Canada, August 11–13, 1993 Proceedings 3*, pages 131–141. Springer, 1993.

3 Hajo Broersma, Ton Kloks, Dieter Kratsch, and Haiko Müller. Independent sets in asteroidal triple-free graphs. *SIAM Journal on Discrete Mathematics*, 12(2):276–287, 1999.

4 Derek G Corneil, Stephan Olariu, and Lorna 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.

5 Derek G Corneil, Stephan Olariu, and Lorna Stewart. Asteroidal triple-free graphs. *SIAM Journal on Discrete Mathematics*, 10(3):399–430, 1997.

6 Bruno Escoffier, Laurent Gourvès, and Jérôme Monnot. Complexity and approximation results for the connected vertex cover problem in graphs and hypergraphs. *Journal of Discrete Algorithms*, 8(1):36–49, 2010.

7 Henning Fernau and David F Manlove. Vertex and edge covers with clustering properties: Complexity and algorithms. *Journal of Discrete Algorithms*, 7(2):149–167, 2009.

8 Michael R Garey and David S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

9 Petr A Golovach, Daniël Paulusma, and Erik Jan van Leeuwen. Induced disjoint paths in at-free graphs. In *Algorithm Theory–SWAT 2012: 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings 13*, pages 153–164. Springer, 2012.

10 Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of generalized vertex cover problems. In *WADS*, pages 36–48. Springer, 2005.

11 Matthew Johnson, Giacomo Paesani, and Daniël Paulusma. Connected vertex cover for $(sp_1 + p_5)$-free graphs. In *Algorithmica82*, pages 20–40, 2020.

12 Dieter Kratsch. Domination and total domination on asteroidal triple-free graphs. *Discrete Applied Mathematics*, 99(1-3):111–123, 2000.

13 Dieter Kratsch, Haiko Müller, and Ioan Todinca. Feedback vertex set on at-free graphs. *Discrete Applied Mathematics*, 156(10):1936–1947, 2008.

14 Daniel Mölle, Stefan Richter, and Peter Rossmanith. Enumerate and expand: New runtime bounds for vertex cover variants. In *Computing and Combinatorics: 12th Annual International Conference, COCOON 2006, Taipei, Taiwan, August 15-18, 2006. Proceedings 12*, pages 265–273. Springer, 2006.

15 Hannes Moser. Exact algorithms for generalizations of vertex cover. *Institut für Informatik, Friedrich-Schiller-Universität Jena*, 12, 2005.

16 Andrea Munaro. Boundary classes for graph problems involving non-local properties. *Theoretical Computer Science*, 692:46–71, 2017.

17 PK Priyadarsini and T Hemalatha. Connected vertex cover in 2-connected planar graph with maximum degree 4 is np-complete. *International Journal of Mathematical, Physical and Engineering Sciences*, 2(1):51–54, 2008.

18 Carla Savage. Depth-first search and the vertex cover problem. *Information processing letters*, 14(5):233–235, 1982.

19 Shuichi Ueno, Yoji Kajitani, and Shin'ya Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1-3):355–360, 1988.

20 Toshimasa Watanabe, Satoshi Kajita, and Kenji Onaga. Vertex covers and connected vertex covers in 3-connected graphs. In *1991 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1017–1020. IEEE, 1991.

On the Fine-Grained Query Complexity of Symmetric Functions

Supartha Podder
Department of Computer Science, Stony Brook University, New York, NY, USA

Penghui Yao
State Key Laboratory for Novel Software Technology, Nanjing University, China
Hefei National Laboratory, China

Zekun Ye
State Key Laboratory for Novel Software Technology, Nanjing University, China

Abstract

Watrous conjectured that the randomized and quantum query complexities of symmetric functions are polynomially equivalent, which was resolved by Ambainis and Aaronson [1], and was later improved in [15, 12]. This paper explores a fine-grained version of the Watrous conjecture, including the randomized and quantum algorithms with success probabilities arbitrarily close to 1/2. Our contributions include the following:

1. An analysis of the optimal success probability of quantum and randomized query algorithms of two fundamental partial symmetric Boolean functions given a fixed number of queries. We prove that for any quantum algorithm computing these two functions using T queries, there exist randomized algorithms using $\mathsf{poly}(T)$ queries that achieve the same success probability as the quantum algorithm, even if the success probability is arbitrarily close to 1/2. These two classes of functions are instrumental in analyzing general symmetric functions.
2. We establish that for any total symmetric Boolean function f, if a quantum algorithm uses T queries to compute f with success probability $1/2+\beta$, then there exists a randomized algorithm using $O(T^2)$ queries to compute f with success probability $1/2+\Omega\left(\delta\beta^2\right)$ on a $1-\delta$ fraction of inputs, where β, δ can be arbitrarily small positive values. As a corollary, we prove a randomized version of Aaronson-Ambainis Conjecture [1] for total symmetric Boolean functions in the regime where the success probability of algorithms can be arbitrarily close to 1/2.
3. We present polynomial equivalences for several fundamental complexity measures of partial symmetric Boolean functions. Specifically, we first prove that for certain partial symmetric Boolean functions, quantum query complexity is at most quadratic in approximate degree for any error arbitrarily close to 1/2. Next, we show exact quantum query complexity is at most quadratic in degree. Additionally, we give the tight bounds of several complexity measures, indicating their polynomial equivalence. Conversely, we exhibit an exponential separation between randomized and exact quantum query complexity for certain partial symmetric Boolean functions.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Query complexity, Symmetric functions, Quantum advantages

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.55

Related Version *Full Version*: `https://arxiv.org/pdf/2309.11279.pdf` [36]

Funding *Supartha Podder*: supported by US National Science Foundation (award no 1954311).
Penghui Yao: supported by National Natural Science Foundation of China (Grant No. 62332009, 61972191) and Innovation Program for Quantum Science and Technology (Grant No. 2021ZD0302900).
Zekun Ye: supported by National Natural Science Foundation of China (Grant No. 62332009, 61972191) and Innovation Program for Quantum Science and Technology (Grant No. 2021ZD0302900).

1 Introduction

Exploring quantum advantages is a key problem in quantum computing. A lot of research work has revolved around analyzing and characterizing quantum advantages, such as [14, 22, 16, 29, 45]. Query complexity is a complexity model commonly used to describe quantum advantages. A comprehensive survey on the query complexity can be found in [24]. A series of works [2, 44, 8, 41] has shown that for partial functions, quantum query complexity could be exponentially smaller (or even less) than the randomized query complexity, while for total functions, they are always polynomially related [4]. Although the query complexity model has demonstrated the powerful ability of a quantum computer to solve certain "structured" problems more efficiently than a classical computer, such as Simon's problem [43] and integer factorization problem [42], there only exist at most quadratic quantum speedups for some "unstructured" problems, such as black-box search problems [23]. Thus, one natural question is to explore how much structure is needed for significant quantum speedups [1].

Watrous conjectured that the randomized query complexity and quantum query complexity of partial symmetric functions are polynomial equivalent [1]. Later, Aaronson and Ambainis [1] initiated the study of quantum speedup on the quantum query complexity of symmetric functions, showing that partial functions invariant under full symmetry do not exhibit super-polynomial quantum speedups, which resolves the Watrous conjecture. Their result was later improved by Chailloux [15], who achieved a tighter bound and removed a technical dependence of output symmetry. Recently [12] performed a systematic analysis of functions symmetric under other group actions and characterized when super-polynomial quantum speedups are achievable. However, all these results work in the bounded error regime and do not explicitly consider arbitrary small biases.

In this paper, we propose and investigate a fine-grained version of the Watrous conjecture concerning the quantum and randomized query complexities of symmetric functions with an arbitrary error. Before stating the conjecture, we need to introduce two notions which are essential to this paper. For any Boolean function f and $T > 0$, let the classical T-bias $\delta_C(f, T)$ be the optimal success probability of T-query randomized algorithms minus $1/2$ over all possible inputs. The quantum T-bias $\delta_Q(f, T)$ is defined for quantum algorithms analogously.

▶ **Conjecture 1** (Fine-grained Watrous conjecture). *There exists a constant $c \geq 2$ satisfying that for any partial symmetric function f and any $T > 0$, $\delta_C(f, c \cdot T) \geq \mathsf{poly}\left(\frac{\delta_Q(f,T)}{T}\right)$.*

The reason for $c \geq 2$ is that the n-bit parity function can be exactly computed with $n/2$ quantum queries, while any randomized algorithm with less n queries succeeds with probability $1/2$. It is also not hard to see the fine-grained Watrous conjecture implies that quantum and randomized query complexities of symmetric functions are polynomially related. Indeed, if $\delta_Q(f, T)$ is lower bounded by some constant, then $\delta_C(f, c \cdot T) \geq \Omega(\frac{1}{\mathsf{poly}(T)})$, which implies the randomized query complexity of f is $\mathsf{poly}(T)$ by error reduction.

1.1 Our Motivation and Contribution

To study the fine-grained Watrous conjecture, we start with the following two fundamental symmetric Boolean functions, which are also essential to analyze general symmetric functions:

$$f_n^k(x) = \begin{cases} 0, & \text{if } |x| \in \{0, n\}, \\ 1, & \text{if } |x| \in \{k, n-k\}, \end{cases}$$

where $1 \leq k \leq n/2$ and

$$f_n^{k,l}(x) = \begin{cases} 0, & \text{if } |x| = k, \\ 1, & \text{if } |x| = l, \end{cases}$$

where $k < l$. The famous Deutsch-Jozsa problem [18] and the decision version of the unstructured search problem [23] can be interpreted as special cases of these two functions. Fefferman and Kimmel considered the subset-sized checking problem [19, 21], where the n-bit input string is promised to have either $\sqrt{n}$ or $0.99\sqrt{n}$ many marked items, and the goal is to decide which case. Their result together with [21] proved that it can be served as an oracle separation between AM and QCMA. On a similar flavour, [5] introduced the approximate counting problem, where the n-bit input has either $\leq w$ or $\geq 2w$ many marked items. Our function $f_n^{k,l}(\cdot)$ can be seen as a symmetric variant of these two problems[1]. We study the tradeoff between the number of queries and the optimal success probabilities in quantum and randomized settings for both functions with errors close to $1/2$.

We further consider the relation between various complexity measures of partial symmetric Boolean functions. For total Boolean functions, it has been proved that several fundamental complexity measures are polynomial equivalent (See Table 1 in [4]). Moreover, the tight bounds on several fundamental complexity measures of total symmetric Boolean functions have also been obtained [24]. However, the result for partial symmetric Boolean functions has not been fully characterized.

Our contribution is as follows. For convenience, if f is a symmetric Boolean function, we denote $f(k) = f(x)$ for any $|x| = k$. Additionally, we say an n-bit Boolean function $f : D \to \{0,1\}$ is even if $f(x) = f(n-x)$ for any $x \in D$, where $D \subseteq \{0,1\}^n$.

1. For both randomized setting and quantum setting, we characterize the optimal success probability of algorithms given the number of queries for the function f_n^k (Theorem 3) and $f_n^{k,l}$ (Theorems 4 and 5). As a corollary, we show for any T-query quantum algorithm to compute f_n^k and $f_n^{k,l}$, there exist classical randomized algorithms using $\mathsf{poly}(T)$ queries to simulate the success probability of the quantum algorithm (Corollaries 6 and 7). Additionally, we characterize the exact quantum query complexity of f_n^k (Theorem 8).
2. We establish a relation between the number of queries and the bias of quantum and randomized algorithms to compute total symmetric Boolean functions, where the bias of the algorithms can be arbitrarily small (Theorem 9). As a corollary, we prove a weak version of Conjecture 2: the acceptance probability of a quantum query algorithm to compute a total symmetric Boolean function can be approximated by a randomized algorithm with only a polynomial increase in the number of queries, where the bias of quantum algorithms can be arbitrarily small (Corollary 10).
3. We investigate the relation between different complexity measures of partial symmetric Boolean functions. Specifically, Theorem 12 shows the relation between the quantum query complexity and the approximate degree of even partial symmetric Boolean functions for arbitrarily small bias[2]. Theorem 13 shows exact quantum query complexity and degree are quadratically related. Theorem 14 presents tight bounds of block sensitivity, fractional block sensitivity, quantum query complexity, and approximate degree, where quantum query complexity and approximate degree are in a bounded-error setting. Corollary 15 shows block sensitivity is an upper bound of quantum query complexity. Since it has

[1] Our problem can also be seen as a Gap-Threshold function. Threshold function is defined as $f_n^k(x) = 1$ iff $|x| \geq k$.

[2] Theorem 12 is also a new result for total symmetric Boolean functions.

been known that $Q(f) \geq \Omega(\sqrt{\mathsf{bs}(f)})$ for any (possibly partial) Boolean function f [11], quantum query complexity and block sensitivity of partial symmetric Boolean functions are polynomially related. On the converse, Theorem 16 shows an exponential gap between the exact quantum query complexity and randomized query complexity for some partial symmetric Boolean functions, which is different from total symmetric functions.

▶ **Conjecture 2** (Aaronson-Ambainis Conjecture [1]). *The acceptance probability of a T-query quantum algorithm to compute a Boolean function can be approximated by a deterministic algorithm using* $\mathrm{poly}(T, 1/\epsilon, 1/\delta)$ *queries within an additive error ϵ on a $1-\delta$ fraction of inputs.*

▶ **Theorem 3.** *For $T > 0$, the quantum T-bias and classical T-bias of f_n^k are*

$$\delta_Q\left(f_n^k, T\right) = \begin{cases} \Theta(\frac{k}{n} \cdot T^2), & \text{if } T \leq \sqrt{n/k}, \\ \Theta\left(1\right), & \text{if } T > \sqrt{n/k}, \end{cases}$$

$$\delta_C\left(f_n^k, T\right) = \begin{cases} 0, & \text{if } T = 1, \\ \Theta(\frac{k}{n} \cdot T), & \text{if } 2 \leq T \leq n/k, \\ \Theta(1), & \text{if } T > n/k. \end{cases}$$

▶ **Theorem 4.** *For $f_n^{k,l}$ and $T > 0$, the quantum T-bias is*

$$\delta_Q\left(f_n^{k,l}, T\right) = \begin{cases} \Theta\left(\min\left\{\frac{l-k}{\sqrt{(n-k)l}} \cdot T, \frac{l-k}{n} \cdot T^2\right\}\right), & \text{if } T = O\left(\frac{\sqrt{(n-k)l}}{l-k}\right), \\ \Theta(1), & \text{if } T = \Omega\left(\frac{\sqrt{(n-k)l}}{l-k}\right). \end{cases}$$

▶ **Theorem 5.** *If $T = O\left(\frac{(n-k)l}{(l-k)^2}\right)$, the classical T-bias of $f_n^{k,l}$ satisfies that*

$$\delta_C\left(f_n^{k,l}, T\right) = O\left(\min\left\{\frac{l-k}{\sqrt{(n-k)l}} \cdot \sqrt{T} + \frac{T}{n}, \frac{l-k}{n} \cdot T\right\}\right),$$

$$\delta_C\left(f_n^{k,l}, T\right) = \Omega\left(\max\left\{\frac{(l-k)^2}{(n-k)l} \cdot T, \frac{l-k}{n} \cdot \sqrt{T}\right\}\right).$$

If $T = \Omega\left(\frac{(n-k)l}{(l-k)^2}\right)$, then $\delta_C\left(f_n^{k,l}, T\right) = \Theta(1)$.

▶ **Corollary 6.** *For arbitrarily small bias $\beta > 0$, if there exists a quantum algorithm using T queries to compute f_n^k with success probability $1/2 + \beta$, then there also exists a classical randomized algorithm using $O(T^2)$ queries to compute f_n^k with the same success probability.*

▶ **Corollary 7.** *For arbitrarily small bias $\beta > 0$, if there exists a quantum algorithm using T queries to compute $f_n^{k,l}$ with success probability $1/2 + \beta$, then there also exist classical randomized algorithms using T^2 queries to compute $f_n^{k,l}$ with success probability $1/2 + \Omega\left(\beta^2\right)$ and using T^4 queries to compute $f_n^{k,l}$ with success probability $1/2 + \Omega\left(\beta\right)$. Thus*

$$\delta_C\left(f_n^{k,l}, T^2\right) \geq \Omega\left(\delta_Q(f_n^{k,l}, T)^2\right) \text{ and } \delta_C\left(f_n^{k,l}, T^4\right) \geq \Omega\left(\delta_Q(f_n^{k,l}, T)\right).$$

▶ **Theorem 8.** *The exact quantum query complexity of f_n^k satisfies $\lceil \frac{\pi}{2\theta} \rceil \leq Q_E(f_n^k) \leq \lceil \frac{\pi}{2\theta} \rceil + 2$, where $\theta = 2 \arcsin \sqrt{k/n}$, whereas the zero-error randomized query complexity of f_n^k is $n-k+1$.*

▶ **Theorem 9.** *For any total symmetric Boolean function f and arbitrarily small bias $\beta > 0$, if there exists a quantum algorithm using T queries to compute f with success probability $1/2 + \beta$, then for any $\delta \in (0,1)$, there exists a randomized algorithm using $O(T^2)$ queries to compute f with success probability $1/2 + \Omega\left(\delta\beta^2\right)$ on a $1-\delta$ fraction of inputs.*

▶ **Corollary 10.** *For any total symmetric Boolean function f and arbitrarily small bias $\beta > 0$, if there exists a T-query quantum algorithm to compute f with success probability $1/2 + \beta$, then for any $\epsilon \in (0, \beta), \delta \in (0,1)$, there exists a randomized algorithm using $O(T^2/(\epsilon^2\delta^2))$ queries to compute f with success probability $1/2 + (\beta - \epsilon)$ on a $1-\delta$ fraction of inputs.*

▶ Remark 11. Corollary 10 is a randomized version of Conjecture 2. Moreover, Corollary 10 considers total symmetric Boolean functions, while Conjecture 2 refers to any Boolean function.

▶ **Theorem 12.** *For any (possibly partial) symmetric Boolean function f satisfying $f(x) = f(n-x)$ and arbitrarily small $\beta > 0$, if $T = \widetilde{\mathsf{deg}}_{\frac{1}{2}-\beta}(f)$, there exists a quantum query algorithm using $\lceil T/2 \rceil$ queries to compute f with success probability $1/2 + \Omega\left(\beta/\sqrt{T}\right)$. Namely,*

$$\delta_Q\left(f, \widetilde{\mathsf{deg}}_{\frac{1}{2}-\beta}(f)\right) = \Omega\left(\frac{\beta}{\sqrt{\widetilde{\mathsf{deg}}_{\frac{1}{2}-\beta}(f)}}\right).$$

As a corollary, we have $Q_\epsilon(f) = O(\widetilde{\mathsf{deg}}_\epsilon(f)^2)$ for any error ϵ arbitrarily close to $1/2$.

▶ **Theorem 13.** *For any partial symmetric Boolean function f, we have $Q_E(f) = O(\mathsf{deg}(f)^2)$.*

▶ **Theorem 14.** *For any partial symmetric Boolean function f, we have*

$$\begin{aligned} \mathsf{bs}(f) = \Theta\left(\mathsf{fbs}(f)\right) &= \left(\max_{k<l:f(k)\neq f(l)} \frac{n}{l-k}\right), \\ Q(f) = \Theta\left(\widetilde{\mathsf{deg}}(f)\right) &= \left(\max_{k<l:f(k)\neq f(l)} \frac{\sqrt{(n-k)l}}{l-k}\right). \end{aligned}$$

▶ **Corollary 15.** *For any partial symmetric Boolean function f, we have $Q(f) = O\left(\mathsf{bs}(f)\right)$.*

▶ **Theorem 16.** *There exists a partial symmetric Boolean function f such that $Q_E(f) = \Omega(n)$ and $R(f) = O(1)$.*

1.2 Proof Techniques

In this section, we give a high-level technical overview of our main results (See full version [36] for the detailed proof).

1.2.1 Upper and Lower Bounds on Quantum T-bias

We use several methods to show the upper bound on the quantum T-bias of different symmetric Boolean functions:

1. For f_n^k, we show if the number of a quantum algorithm is no more than T queries, then the bias β of the algorithm is at most $O(T^2/\,\mathsf{bs}(f_n^k))$, where $\mathsf{bs}(f_n^k)$ is the block sensitivity of f_n^k. By solving a lower bound of $\mathsf{bs}(f_n^k)$, we obtain an upper bound on the quantum T-bias of f_n^k (Theorem 3).
2. For $f_n^{k,l}$, using Paturi's lower bound technique [35] for the approximate degree of symmetric Boolean functions, we give the following lower bound:

$$Q_\epsilon(f_n^{k,l}) \geq \frac{1}{2}\widetilde{\mathsf{deg}}_\epsilon\left(f_n^{k,l}\right) = \Omega\left(\max\left\{\frac{\beta\sqrt{(n-k)\,l}}{l-k}, \sqrt{\frac{\beta n}{l-k}}\right\}\right),$$

where $\beta = 1/2-\epsilon$. The quantum T-bias of $f_n^{k,l}$ is derived by this lower bound (Theorem 4).

To obtain the lower bound on the quantum T-bias, we also use diverse ideas to design T-query quantum algorithms:

1. For f_n^k and $f_n^{k,l}$, we use various variants of amplitude amplification algorithm and analyze the success probability of algorithms meticulously (Theorems 3 and 4).
2. For even symmetric Boolean functions, we design a novel quantum algorithm by taking advantage of the Chebyshev expansion and constructing controlled Grover's diffusion operations (Theorem 12).

1.2.2 Upper and Lower Bounds on Classical T-bias

For f_n^k and $f_n^{k,l}$, we show the upper bound on the classical T-bias by analyzing the total variation distance of distributions; for the lower bound, we give sampling algorithms to estimate Hamming weights of the input and analyze the success probability of the algorithms also by analyzing the distance between distributions (Theorems 3 and 5).

For the lower bound on the classical T-bias of total symmetric Boolean functions, we design an innovative randomized algorithm by utilizing the Kravchuk polynomial when the number of queries is T. The analysis of the algorithm also uses the orthogonality property of the Kravchuk polynomial (Theorem 9).

1.2.3 The Relation Between Complexity Measures

The key ideas to build the relation between complexity measures of partial symmetric Boolean functions are as follows:

1. In Theorem 13, we show the relation between the exact quantum query complexity and the degree by giving the lower bound of the degree and designing a matching exact quantum algorithm up to a polynomial level. Similar to the proof of Theorem 8, the exact quantum algorithm makes use of a subroutine to distinguish $|x| = k$ from $|x| = l$ exactly [25].
2. In Theorem 14, the analysis of block sensitivity and fractional block sensitivity relies on the symmetry property of the function. Furthermore, we show the quantum query complexity and the approximate degree of any partial symmetric Boolean function f are equivalent to a constant factor. While the lower bound is well known (Fact 1), we show $Q(f) \leq \widetilde{\mathsf{deg}}(f)$ by giving a quantum approximate counting algorithm using $O(\widetilde{\mathsf{deg}}(f))$ quantum queries.
3. The exponential gap in Theorem 16 is shown by giving a function easy to compute in a bounded-error case but has a large degree.

1.3 Related Work

The need for structure in quantum speedups has been studied extensively. Beals, Buhrman, Cleve, Mosca and de Wolf [9] showed that there exists at most polynomial quantum speedups for total Boolean functions in the query model. Thus, the exponential speedups may only occur at partial functions. Furthermore, Aaronson and Ambainis [1] showed that symmetric functions do not allow super-polynomial quantum speedups, even if the functions are partial. Chailloux [15] improved this result for a broader class of symmetric functions. Ben-David, Childs, Gilyén, Kretschmer, Podder and Wang [12] further showed that hypergraph symmetries in the adjacency matrix model allow at most polynomial separations between quantum and randomized query complexities. Ben-David [10] proved a classical and quantum polynomial equivalence for a class of functions satisfying a certain symmetric promise. Aaronson and Ben-David [3] showed that there exists at most polynomial quantum speedups to compute an n-bit partial Boolean function if the domain $D = \text{poly}(n)$. Nonetheless, all these results concern the algorithms with a constant probability of success. They do not cover the query complexity with a subconstant probability of success.

We also survey some results about the optimal success probability of quantum algorithms when the number of queries is fixed. For the unstructured search problem, Zalka [46] showed an optimal success probability of a quantum algorithm given the number of queries. For the collision finding problem, Zhandry [47] gave the upper bound on the success probability of quantum algorithms when the number of queries is fixed, which matched the algorithm proposed by Brassard, Høyer and Tapp [13]. Ambainis and Iraids [6] analyzed the optimal success probability of one-query quantum algorithms to compute EQUALITY_n and AND_n functions. Montanaro, Jozsa, and Mitchison [33] indicated the optimal success probability of small symmetric Boolean functions when given any number of queries by numerical results. There is not much study about the optimal success probability with a given number of queries for symmetric Boolean functions. Our work will fill the gap in this field.

For the complexity measures of a nonconstant n-bit total symmetric Boolean function f, it has been known that $R(f)$, $D(f)$, $\mathsf{deg}(f)$, $s(f)$, $bs(f)$ are $\Theta(n)$, and $Q(f) = \Theta\left(\widetilde{\mathsf{deg}}(f)\right) = \Theta\left(\sqrt{n(n-\Gamma(f))}\right)$, where $\Gamma(f) = \min\{|2k-n+1| : f(k) \neq f(k+1)\}$ [24]. Sherstov [40] gave an almost tight characterization of $\mathsf{deg}_\epsilon(f)$ for specific $\epsilon \in [1/2^n, 1/3]$. Afterward, de Wolf [17] obtained the optimal bound. Regarding the complexity measures of partial symmetric Boolean functions, Aaronson and Ambainis [1] showed for any partial symmetric Boolean function f, $R(f) = O\left(Q(f)^2\right)$ as mentioned before. Researchers also studied the exact quantum query complexity for many instances of partial symmetric Boolean functions. For example, Deutsch and Jozsa [18] studied the first partially symmetric Boolean function. Afterward, generalized Deutsch-Jozsa problems were studied in [33, 37, 38]. He, Sun, Yang and Yuan [25] established the asymptotically optimal bound for the exact quantum query complexity of distinguishing whether $|x| = k$ or l. Qiu and Zheng [37, 39] studied the exact quantum query complexity of symmetric Boolean functions with degree 1 or 2. Additionally, several works [7, 20, 31] explored the connections between block sensitivity, fractional block sensitivity and degree for bounded functions.

In a similar work, Montanaro, Nishimura and Raymond [34] studied the unbounded error query complexity of Boolean functions in a scenario where it is only required that the query algorithm succeeds with a probability strictly greater than 1/2. They proved quantum and classical query complexities are related by a constant factor for any (possibly partial) Boolean function. Similar results are also known in the communication complexity model [27, 26]. Compared to the result in [34], we aim to analyze the relation between

quantum/classical query complexity and bias more precisely. For instance, we show for any quantum algorithm computing f_n^k and $f_n^{k,l}$ using T queries, there exist randomized algorithms using $\mathrm{poly}(T)$ queries that have the same bias as the quantum algorithm. Such a conclusion is not implied by [34] since the unbounded error model only requires a strictly positive bias without quantitative analysis.

1.4 Organization

The remainder of the paper is organized as follows. In Section 2, we review some definitions and facts. In Section 3, we prove Theorem 9 pertaining to connections between quantum and randomized algorithms of symmetric Boolean functions in the small-bias regime. In Section 4, we prove Theorem 12 to show the relation between the quantum query complexity and the approximate degree for arbitrarily small bias. Finally, a conclusion is made in Section 5.

2 Preliminaries

For an n-bit Boolean function $f : D \to \{0,1\}$, if $D = \{0,1\}^n$, f is a total function; if $D \subset \{0,1\}^n$, f is a partial function. We say f is symmetric if $f(x)$ only depends on $|x|$, where $|x|$ is the number of 1's in x. Correspondingly, we say $g : \{-1,1\}^n \to \mathbb{R}$ is symmetric if $g(x)$ only depends on $|x|$, where $|x|$ is the number of -1's in x. Every $g : \{-1,1\}^n \to \mathbb{R}$ can be uniquely expressed as $g(x) = \sum_{S\subseteq[n]} \widehat{g}(S)x_S$, where $x_S = \prod_{j\in S} x_j$ and $\widehat{g}(S)$ is the Fourier coefficient of g for any $S \subseteq [n]$. Let $H(n,i,T)$ be the hypergeometric distribution sampling T times from $x \in \{0,1\}^n$ satisfying that $|x| = i$ without replacement. A binomial distribution with parameters n, p is written as $B(n,p)$.

2.1 Query Models and Complexity Measures

In the classical query model, for an input $x \in \{0,1\}^n$, we can obtain x_i for some i by making one query. The deterministic query complexity of f, denoted by $D(f)$, is the minimum number of queries required by a deterministic algorithm to compute f on the worst input. The randomized query complexity of f, denoted by $R_\epsilon(f)$, is the minimum number of queries required by a randomized algorithm to compute f with error ϵ on the worst input. If $\epsilon = 1/3$, we abbreviate $R_\epsilon(f)$ to $R(f)$. Moreover, $R_0(f)$ is called the zero-error randomized query complexity of f.

In the quantum query model, a query algorithm can be described as follows: it starts with a fixed state $|\psi_0\rangle$ and then performs the sequence of operations $U_0, O_x, U_1, \ldots, O_x, U_t$, where U_i's are unitary operators not depend on x and the query oracle O_x is defined as $O_x|i\rangle|b\rangle = |i\rangle|x_i \oplus b\rangle$ for any $i \in [n]$ and $b \in \{0,1\}$. This leads to the final state $|\psi_x\rangle = U_t O_x U_{t-1} \cdots U_1 O_x U_0 |\psi_0\rangle$. The output result is obtained by measuring $|\psi_x\rangle$. The exact query complexity of f, denoted by $Q_E(f)$, is the minimum number of queries required by a quantum algorithm to compute f exactly on the worst input. Such a quantum algorithm is called an exact quantum algorithm. The quantum query complexity of f, denoted by $Q_\epsilon(f)$, is the minimum number of queries required by a quantum algorithm to compute f with ϵ on the worst input. If $\epsilon = 1/3$, we abbreviate $Q_\epsilon(f)$ to $Q(f)$.

Then we overview some notations about complexity measures of Boolean functions. The degree of f, denoted as $\mathsf{deg}(f)$, is the minimum degree of all real multilinear polynomial representations of f. The approximate degree of f, denoted by $\widetilde{\mathsf{deg}}_\epsilon(f)$, is the minimum degree among all real multilinear polynomials that approximate f with error ϵ. If $\epsilon = 1/3$, we abbreviate $\widetilde{\mathsf{deg}}_\epsilon(f)$ as $\widetilde{\mathsf{deg}}(f)$. The block sensitivity of f on x, denoted as $\mathsf{bs}(f,x)$, is the

maximum number of disjoint sensitive blocks in x. The block sensitivity of f is defined as $\mathsf{bs}(f) = \max_x \mathsf{bs}(f,x)$. The value of $\mathsf{bs}(f,x)$ can be expressed as an integer linear program. The fractional relaxation of the integer program yields the fractional block sensitivity of f on x, denoted as $\mathsf{fbs}(f,x)$. The fractional block sensitivity of f is defined as $\mathsf{fbs}(f) = \max_x \mathsf{fbs}(f,x)$.

▶ **Fact 1** ([9]). *If f is a Boolean function, then $Q_E(f) \geq \mathsf{deg}(f)/2$ and $Q_\epsilon(f) \geq \widetilde{\mathsf{deg}}_\epsilon(f)/2$.*

2.2 Orthonormal Polynomials and Fourier Growth

▶ **Fact 2** (Corollary 2.3 in [30]). *For any $0 \leq j \leq T$, the Kravchuk polynomial is defined as*

$$K_j(t,T) = \sum_{i=0}^{j} \binom{t}{i}\binom{T-t}{j-i}(-1)^i.$$

Then for any $0 \leq l, m \leq T$, there exists the following orthogonality property:

$$\sum_{t=0}^{T} \binom{T}{t} K_l(t,T)K_m(t,T) = 2^T \binom{T}{l}\delta_{l,m},$$

where $\delta_{l,m} = 1$ if $l = m$, and $\delta_{l,m} = 0$ if $l \neq m$.

▶ **Fact 3** (Parseval's identity, Page 84 in [32]). *For a function $g : [-1,1] \to [-1,1]$, if $g(x) = \sum_{i=0}^{T} a_i T_i(x)$ for any $x \in [-1,1]$, where T_i is the Chebyshev polynomial such that $T_i(\cos\theta) = \cos(i\theta)$, then*

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} \left(g(x)\right)^2 dx = \pi a_0^2 + \frac{\pi}{2}\sum_{i=1}^{T} a_i^2.$$

▶ **Fact 4** (Theorem 1 in [28]). *If symmetric function $f : \{-1,1\}^n \to [-1,1]$ has degree d, then*

$$\sum_{S\subseteq[n]:|S|=l} |\widehat{f}(S)| \leq \frac{d^l}{l!},$$

where $\widehat{f}(S)$ is the Fourier coefficients of f for any $S \subseteq [n]$.

3 The Relation Between Quantum and Randomized Algorithms of Symmetric Boolean Functions for Arbitrarily Small Bias

In this section, we give the proof of Theorem 9. First, we state Lemma 17, which is needed to prove the theorem. In Lemma 17, since g is a symmetric function, we let $\widehat{g}(l) = \widehat{g}(S)$ for any $|S| = l$ with a slight abuse of notation, where $\widehat{g}(S)$ is the Fourier coefficients of g for $S \subseteq [n]$. Moreover, $K_l(t,T)$ is the Kravchuk polynomial as Fact 2.

▶ **Lemma 17.** *Given a symmetric function $g : \{-1,1\}^n \to [-1,1]$ such that $\mathsf{deg}(g) = d$, for any $d \leq T \leq n$ and $x \in \{-1,1\}^n$, we have*

$$g(x) = \mathbb{E}_{t\sim H(n,|x|,T)} \sum_{l=0}^{d} \widehat{g}(l)\binom{n}{l}\frac{K_l(t,T)}{\binom{T}{l}},$$

$$\mathbb{E}_{t\sim B(T,\frac{1}{2})} \left(\sum_{l=0}^{d} \widehat{g}(l)\binom{n}{l}\frac{K_l(t,T)}{\binom{T}{l}}\right)^2 \leq 2.$$

Proof. Since $g : \{-1,1\}^n \to [-1,1]$ is a symmetric function and $\mathsf{deg}(g) = d$, for any $d \le T \le n$ and $x \in \{-1,1\}^n$, we have

$$\begin{aligned}
g(x) &= \sum_{S\subseteq[n]:|S|\le d} \widehat{g}(S)x_S \\
&= \sum_{l=0}^{d} \widehat{g}(l) \sum_{S\subseteq[n]:|S|=l} x_S \\
&= \sum_{l=0}^{d} \widehat{g}(l) \frac{1}{\binom{n-l}{T-l}} \sum_{U\subseteq[n]:|U|=T} \sum_{S\subseteq U:|S|=l} x_S \\
&= \sum_{l=0}^{d} \widehat{g}(l) \frac{\binom{n}{l}}{\binom{n}{T}\binom{T}{l}} \sum_{U\subseteq[n]:|U|=T} \sum_{S\subseteq U:|S|=l} x_S \\
&= \frac{1}{\binom{n}{T}} \sum_{U\subseteq[n]:|U|=T} \sum_{l=0}^{d} \widehat{g}(l)\binom{n}{l} \frac{\sum_{S\subseteq U:|S|=l} x_S}{\binom{T}{l}} \\
&= \frac{1}{\binom{n}{T}} \sum_{t=0}^{T} \binom{|x|}{t}\binom{n-|x|}{T-t} \sum_{l=0}^{d} \widehat{g}(l)\binom{n}{l} \frac{\sum_{i=0}^{l}\binom{t}{i}\binom{T-t}{l-i}(-1)^i}{\binom{T}{l}} \\
&= \mathbb{E}_{t\sim H(n,|x|,T)} \sum_{l=0}^{d} \widehat{g}(l)\binom{n}{l} \frac{K_l(t,T)}{\binom{T}{l}}.
\end{aligned}$$

Let $c_l = \widehat{g}(l)\binom{n}{l}$. By Fact 4, we have $|c_l| \le \frac{d^l}{l!}$. Then we have

$$\begin{aligned}
\mathbb{E}_{t\sim B(T,\frac{1}{2})} \left(\sum_{l=0}^{d} \widehat{g}(l)\binom{n}{l}\frac{K_l(t,T)}{\binom{T}{l}}\right)^2 &= \frac{1}{2^T}\sum_{t=0}^{T}\binom{T}{t}\left(\sum_{l=0}^{d} c_l \frac{K_l(t,T)}{\binom{T}{l}}\right)^2 \\
&= \sum_{l=0}^{d} \frac{c_l^2}{\binom{T}{l}} \\
&\le \sum_{l=0}^{d} \frac{d^l}{l!}\cdot\frac{d^l}{l!}\cdot\frac{l!}{\prod_{i=0}^{l-1} T-i} \\
&\le \sum_{l=0}^{d} \frac{1}{l!}\cdot\frac{d^{2l}}{\prod_{i=0}^{l-1} T-i} \\
&\le \sum_{l=0}^{d}\prod_{i=0}^{l-1}\frac{d^2}{T-i} \\
&\le \sum_{l=0}^{d}\left(\frac{1}{2}\right)^l \\
&\le 2,
\end{aligned}$$

where the second equality comes from the orthogonality property of the Kravchuk polynomial (Fact 2). ◀

Proof of Theorem 9. For any total symmetric Boolean function f and $0 < \beta < 1/2, 0 < \delta < 1$, let $\epsilon = 1/2 - \beta$. Suppose there exists a quantum algorithm using T queries to compute f with success probability $1/2+\beta$. By Fact 1, we have $\widetilde{\mathsf{deg}}_\epsilon(f) \le 2T$. Let $d = \widetilde{\mathsf{deg}}_\epsilon(f)$. Next, it suffices to prove there exists a randomized algorithm using $O(d^2)$ queries to compute f with success probability $1/2 + \Omega\left(\delta\beta^2\right)$ on a $1-\delta$ fraction of inputs.

Since $d = \widetilde{\mathsf{deg}}_\epsilon(f)$, there exists a degree-$d$ symmetric function $f' : \{0,1\}^n \to [0,1]$ satisfying if $f(x) = 0$, then $f'(x) \leq 1/2 - \beta$; if $f(x) = 1$, then $f'(x) \geq 1/2 + \beta$. It means that $(1 - 2f(x))(1 - 2f'(x)) \geq 2\beta$. Let $h : \{0,1\}^n \to [-1,1]$ be defined as $h(x) = 1 - 2f'(x)$ and $g : \{-1,1\}^n \to [-1,1]$ defined as $g(1-2x) = h(x)$ for any $x \in \{0,1\}^n$. Then g is also a degree-d symmetric function. By Lemma 17, for any $d \leq T \leq n$ and $x \in \{-1,1\}^n$, we have

$$g(x) = \mathbb{E}_{t\sim H(n,|x|,T)} \sum_{l=0}^{d} \widehat{g}(l) \binom{n}{l} \frac{K_l(t,T)}{\binom{T}{l}}.$$

Let

$$A_t = \sum_{l=0}^{d} \widehat{g}(l) \binom{n}{l} \frac{K_l(t,T)}{\binom{T}{l}}. \tag{1}$$

Then for $x \in \{0,1\}^n$, we have $h(x) = \mathbb{E}_{t\sim H(n,|x|,T)} A_t$, where $|x|$ is the number of 1's in x. For any $0 \leq t \leq T$, let

$$A'_t = \begin{cases} \min\left\{A_t, \frac{16}{\delta\beta}\right\}, & \text{if } A_t \geq 0, \\ \max\left\{A_t, -\frac{16}{\delta\beta}\right\}, & \text{if } A_t < 0. \end{cases} \tag{2}$$

Suppose x follows the uniform distribution of $\{0,1\}^n$. We give Algorithm 1 to compute $f(x)$ using $T = 2d^2 + d$ queries. The error analysis of Algorithm 1 is as follows. Given

Algorithm 1 A T-query quantum algorithm to compute $f(x)$.

1. Query T distinct bits in x uniformly and denote the number of 1's by t.
2. Compute the value of A_t as Equation (1).
3. Output 0 with the probability $\frac{1}{2}(1 + \frac{\delta\beta}{16}A'_t)$ and output 1 with the probability $\frac{1}{2}(1 - \frac{\delta\beta}{16}A'_t)$, where A'_t is defined as Equation (2).

$x \in \{0,1\}^n$, let $h'(x) = \mathbb{E}_{t\sim H(n,|x|,T)} A'_t$. Then the probability that the algorithm outputs 0 is $\frac{1}{2}\left(1 + \frac{\delta\beta}{16}h'(x)\right)$ and the probability that the algorithm outputs 1 is $\frac{1}{2}\left(1 - \frac{\delta\beta}{16}h'(x)\right)$. By Lemma 17, we have $\mathbb{E}_{t\sim B(T,\frac{1}{2})} A_t^2 \leq 2$. Since

$$\left|\mathbb{E}_{t\sim B(T,\frac{1}{2})} A_t\right| = \left|\frac{1}{2^n} \sum_{x\in\{0,1\}^n} \mathbb{E}_{t\sim H(n,|x|,T)} A_t\right| = \left|\frac{1}{2^n} \sum_{x\in\{0,1\}^n} h(x)\right| \leq 1, \tag{3}$$

we have $\sigma^2(A_t) = \mathbb{E}A_t^2 - (\mathbb{E}A_t)^2 \leq 2$ when t follows the binomial distribution $B(T, \frac{1}{2})$. By Chebyshev's inequality, we have

$$P(|A_t - \mathbb{E}A_t| \geq 2\delta) \leq \frac{1}{\delta^2}. \tag{4}$$

By Equation (2), we have

$$A'_t = \begin{cases} A_t, & \text{if } |A_t| \leq \frac{16}{\delta\beta}. \\ -\frac{16}{\delta\beta}, & \text{if } A_t < -\frac{16}{\delta\beta}. \\ \frac{16}{\delta\beta}, & \text{if } A_t > \frac{16}{\delta\beta}. \end{cases}$$

Thus if $|A_t| \leq \frac{16}{\delta\beta}$, then $|A_t - A'_t| = 0$; if $|A_t| > \frac{16}{\delta\beta}$, then $|A_t - A'_t| = |A_t| - \frac{16}{\delta\beta}$. Then we have

$$\mathbb{E}_t|A_t - A'_t| = \mathbb{E}_{t:|A_t|\geq\frac{16}{\delta\beta}}\left(|A_t| - \frac{16}{\delta\beta}\right). \tag{5}$$

By Equation (3), we have $|\mathbb{E}A_t| \leq 1$. Since $0 < \delta < 1, 0 < \beta < 1/2$, if $|A_t| > \frac{16}{\delta\beta}$, then $|A_t - \mathbb{E}A_t| \geq |A_t| - 1 \geq \frac{15}{\delta\beta}$. Thus, we have

$$\begin{aligned}
\mathbb{E}_{t:|A_t|\geq\frac{16}{\delta\beta}}\left(|A_t| - \frac{16}{\delta\beta}\right) &\leq \mathbb{E}_{t:|A_t-\mathbb{E}A_t|\geq\frac{15}{\delta\beta}}\left(|A_t| - \frac{16}{\delta\beta}\right) \\
&= \sum_{a=0}^{\infty}\mathbb{E}_{t:\frac{15\cdot 2^a}{\delta\beta}\leq|A_t-\mathbb{E}A_t|\leq\frac{15\cdot 2^{a+1}}{\delta\beta}}\left(|A_t| - \frac{16}{\delta\beta}\right) \\
&\leq \sum_{a=0}^{\infty}\mathbb{E}_{t:|A_t-\mathbb{E}A_t|\geq\frac{15\cdot 2^a}{\delta\beta}}\left(\frac{15\cdot 2^{a+1}}{\delta\beta} + 1 - \frac{16}{\delta\beta}\right) \\
&\leq \sum_{a=0}^{\infty}\mathbb{E}_{t:|A_t-\mathbb{E}A_t|\geq\frac{15\cdot 2^a}{\delta\beta}}\frac{16\left(2^{a+1}-1\right)}{\delta\beta} \\
&\leq \sum_{a=0}^{\infty}\left(\frac{\delta\beta}{15\cdot 2^{a-1}}\right)^2\cdot\frac{16\left(2^{a+1}-1\right)}{\delta\beta} \\
&= \frac{64}{225}\delta\beta\cdot\sum_{a=0}^{\infty}\frac{2^{a+1}-1}{4^a} \\
&\leq \delta\beta,
\end{aligned} \tag{6}$$

where the fourth inequality comes from Equation (4). Combining Equations (5) and (6), we have

$$\mathbb{E}_{t\sim B(T,\frac{1}{2})}|A_t - A'_t| \leq \delta\beta. \tag{7}$$

For $x \in \{0,1\}^n$, we have $h(x) = \mathbb{E}_{t\sim H(n,|x|,T)}A_t$ and $h'(x) = \mathbb{E}_{t\sim H(n,|x|,T)}A'_t$. Then we have

$$\frac{1}{2^n}\sum_{x\in\{0,1\}^n}|h(x) - h'(x)| = \frac{1}{2^n}\sum_{x\in\{0,1\}^n}\mathbb{E}_{t\sim H(n,|x|,T)}|A_t - A'_t| = \mathbb{E}_{t\sim B(T,\frac{1}{2})}|A_t - A'_t| \leq \delta\beta.$$

by Equation 7. Thus, there are at least $1-\delta$ fractions of inputs x such that $|h(x)-h'(x)| \leq \beta$. For such x, since $h(x)(1-2f(x)) \geq 2\beta$, we have $h'(x)(1-2f(x)) \geq \beta$. Therefore, if $f(x) = 0$, then $h'(x) \geq \beta$; if $f(x) = 1$, then $h'(x) \leq -\beta$. Thus, for at least $1-\beta$ fractions of inputs, the bias of the algorithm is at least $\beta\cdot\delta\beta/16 = \delta\beta^2/16$. ◀

4 The Relation Between Quantum Query Complexity and Approximate Degree for Arbitrarily Small Bias

In this section, we give the proof of Theorem 12.

Proof of Theorem 12. Given a (possibly partial) n-bit symmetric Boolean function $f : D \to \{0,1\}$, where $D \subseteq \{0,1\}^n$ and $f(x) = f(n-x)$ for any $x \in D$. For $0 < \epsilon < 1/2$, let $T = \mathsf{deg}_\epsilon(f)$ and $\beta = 1/2 - \epsilon$. Same as the proof of Theorem 9, for any function f' that approximates f with error ϵ, we have $(1-2f(x))(1-2f'(x)) \geq 2\beta$.

Let $g : [-1,1] \to [-1,1]$ be defined as $g\left(1-2|x|/n\right) = 1-2f(x)$ for any $x \in D$. Since $f(x) = f(n-x)$, g is an even function. Assume function $h : [-1,1] \to [-1,1]$ is the optimal approximation polynomial of g with degree T. Then $g(x)h(x) \geq 2\beta$ and h is also an

even function. Thus, $h(x)$ can be expressed as $\sum_{i=0}^{\lceil T/2\rceil} a_i T_{2i}(x)$ for any $x \in [-1,1]$, where $T_{2i}(x)$ is the Chebyshev polynomial of degree $2i$ and $T_{2i}(\cos\eta) = \cos 2i\eta$ for any $\eta \in [0,\pi]$. Furthermore, we have $h(\cos\eta) = \sum_{i=0}^{\lceil T/2\rceil} a_i \cos 2i\eta$ for any $\eta \in [0,\pi]$. Let $\cos\eta_x = 1 - 2|x|/n$. Then $(1-2f(x))\, h(\cos\eta_x) \geq 2\beta$ and

$$\begin{aligned} h(\cos\eta_x) &= \sum_{i=0}^{\lceil T/2\rceil} a_i \cos 2i\eta_x \\ &= \sum_{i:a_i\geq 0} a_i \left(2\cos^2 i\eta_x - 1\right) + \sum_{i:a_i<0} a_i \left(1 - 2\sin^2 i\eta_x\right) \\ &= \left(\sum_{i:a_i\geq 0} 2a_i \cos^2 i\eta_x - \sum_{i:a_i<0} 2a_i \sin^2 i\eta_x\right) + \left(\sum_{i:a_i<0} a_i - \sum_{i:a_i\geq 0} a_i\right) \\ &= \Delta_x - M, \end{aligned} \tag{8}$$

where $\Delta_x = 2\left(\sum_{a_i\geq 0} a_i \cos^2 i\eta_x - \sum_{a_i<0} a_i \sin^2 i\eta_x\right)$ and $M = \sum_{i=0}^{\lceil T/2\rceil} |a_i|$. By Fact 3, $\sum_{i=0}^{\lceil T/2\rceil} a_i^2 \leq \frac{2}{\pi}\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} dx = 2$. Thus,

$$M \leq \sqrt{2\lceil T/2\rceil + 1} \leq \sqrt{2(T+1)}. \tag{9}$$

Let $|\psi\rangle = \frac{1}{\sqrt{n}} \sum_{i\in[n]} |i\rangle |-\rangle$. Then $\langle\psi|O_x|\psi\rangle = 1 - 2|x|/n = \cos\eta_x$. As a result, there exists a state $|\psi^\perp\rangle$ such that $\langle\psi|\psi^\perp\rangle = 0$ and $O_x|\psi\rangle = \cos\eta_x |\psi\rangle + \sin\eta_x |\psi^\perp\rangle$. For the following reflection operation

$$S_0 = 2|\psi\rangle\langle\psi| - I, S_1 = 2O_x|\psi\rangle\langle\psi| O_x - I = O_x S_0 O_x, \tag{10}$$

we have

$$\begin{aligned} S_1 S_0 |\psi\rangle &= \cos 2\eta_x |\psi\rangle + \sin 2\eta_x |\psi^\perp\rangle, \\ S_1 S_0 |\psi^\perp\rangle &= -\sin 2\eta_x |\psi\rangle + \cos 2\eta_x |\psi^\perp\rangle. \end{aligned} \tag{11}$$

Let R_0 be the corresponding controlled operation of S_0, i.e., for any $|\phi\rangle$,

$$R_0|\phi\rangle|+\rangle = |\phi\rangle|+\rangle, R_0|\phi\rangle|-\rangle = (S_0|\phi\rangle)|-\rangle. \tag{12}$$

Let $|\pm_i\rangle = \underbrace{|-\rangle\cdots|-\rangle}_{i}\underbrace{|+\rangle\cdots|+\rangle}_{\lceil T/2\rceil - i}$. If $a_i \geq 0$, let

$$P_i^+ = (|\psi\rangle\langle\psi|)\otimes(|\pm_i\rangle\langle\pm_i|), P_i^- = (I - |\psi\rangle\langle\psi|)\otimes(|\pm_i\rangle\langle\pm_i|).$$

If $a_i < 0$, let

$$P_i^- = (|\psi\rangle\langle\psi|)\otimes(|\pm_i\rangle\langle\pm_i|), P_i^+ = (I - |\psi\rangle\langle\psi|)\otimes(|\pm_i\rangle\langle\pm_i|).$$

Let $P_0 = \sum_i P_i^+, P_1 = \sum_i P_i^-$. Then $P_0 + P_1 = I$. Let $\alpha_i = \sqrt{\frac{|a_i|}{M}}$. Then $\sum_i \alpha_i^2 = 1$. We give Algorithm 2 to compute $f(x)$ and analyze the success probability of the algorithm as follows. Since R_0 is the corresponding controlled reflection operation of S_0, the final state after performing Step 2 of Algorithm 2 is

$$\sum_{i=0}^{\lceil T/2\rceil} \alpha_i \left(\underbrace{(O_x S_0)\cdots(O_x S_0)}_{i \text{ times}}|\psi\rangle\right)|\pm_i\rangle.$$

Algorithm 2 A T-query quantum algorithm to compute $f(x)$.

1. Prepare the initial state $\sum_{i=0}^{\lceil T/2\rceil} \alpha_i |\psi\rangle |\pm_i\rangle$, which consists of the first qudit and $\lceil T/2\rceil$ ancillary qubits, where $\alpha_i, |\psi\rangle, |\pm_i\rangle$ are defined on Page 11.
2. For $i = 1$ to $\lceil T/2\rceil$, we perform unitary operation $(O_x \otimes I)R_0$ in the first qudit and the i-th ancillary qubit, where R_0 is given in Equation (12).
3. Perform the project measurement $\{P_0, P_1\}$ defined on Page 11 to the final state and output the measurement result.

If i is even, then

$$\begin{aligned}\left(\underbrace{(O_xS_0)\cdots(O_xS_0)}_{i \text{ times}}|\psi\rangle\right)|\pm_i\rangle &= \left(\underbrace{(O_xS_0O_xS_0)\cdots(O_xS_0O_xS_0)}_{i/2 \text{ times}}|\psi\rangle\right)|\pm_i\rangle \\ &= \left(\underbrace{(S_1S_0)\cdots(S_1S_0)}_{i/2 \text{ times}}|\psi\rangle\right)|\pm_i\rangle \\ &= \left(\cos i\eta_x |\psi\rangle + \sin i\eta_x |\psi^\perp\rangle\right)|\pm_i\rangle,\end{aligned}$$

where the second equality comes from $S_1 = O_xS_0O_x$ and the third equality comes from Equation (11). Similarly, if i is odd, by Equation (11) and $S_1 = O_xS_0O_x$, we have

$$\begin{aligned}\left(\underbrace{(O_xS_0)\cdots(O_xS_0)}_{i \text{ times}}|\psi\rangle\right)|\pm_i\rangle &= \left(\underbrace{(O_xS_0)\cdots(O_xS_0)}_{i-1 \text{ times}}\left(\cos\eta_x |\psi\rangle + \sin\eta_x |\psi^\perp\rangle\right)\right)|\pm_i\rangle \\ &= \left(\underbrace{(S_1S_0)\cdots(S_1S_0)}_{(i-1)/2}\left(\cos\eta_x |\psi\rangle + \sin\eta_x |\psi^\perp\rangle\right)\right)|\pm_i\rangle \\ &= \left(\cos i\eta_x |\psi\rangle + \sin i\eta_x |\psi^\perp\rangle\right)|\pm_i\rangle.\end{aligned}$$

Thus, after performing Step 2 of Algorithm 2, the final state is

$$\sum_{i=0}^{\lceil T/2\rceil} \alpha_i \left(\cos i\eta_x |\psi\rangle + \sin i\eta_x |\psi^\perp\rangle\right)|\pm_i\rangle.$$

By Equation (8), the probability that the measurement result is 0 is

$$\begin{aligned}p_x &= \sum_{i:a_i\geq 0} \alpha_i^2 \cos^2 i\eta_x + \sum_{i:a_i<0} \alpha_i^2 \sin^2 i\eta_x \\ &= \frac{1}{M}\left(\sum_{i:a_i\geq 0} a_i \cos^2 i\eta_x - \sum_{i:a_i<0} a_i \sin^2 i\eta_x\right) \\ &= \frac{\Delta_x}{2M} \\ &= \frac{1}{2} + \frac{h(\cos\eta_x)}{2M},\end{aligned}$$

and the probability that the algorithm outputs 1 is $1/2 - h(\cos\eta_x)/(2M)$. Since $(1 - 2f(x))h(\cos\eta_x) \geq 2\beta$, the probability that the algorithm outputs $f(x)$ is at least $1/2 + \beta/M$.

By Equation (9), the bias of the algorithm is at least $\beta/M \geq \beta/\sqrt{2T+2}$. Then we can amplify the success probability to $1/2+\beta$ by running $O(T)$ times Algorithm 2 repetitively (see full version [36] for the detailed proof). Thus, there exists a quantum algorithm using $O(T^2)$ queries to with success probability $1-\epsilon$, which implies $Q_\epsilon(f) = O\left(\widetilde{\mathsf{deg}}_\epsilon(f)^2\right)$. ◀

▶ Remark 18. We conjecture that $f(x) = f(n-x)$ is not a necessary condition. If an n-bit (possibly partial) symmetric Boolean function f satisfies that $f(x) \neq f(n-x)$ for some $x \in D$, we can define a new $2n$-bit Boolean function f^* such that

$$f^*(x) = \begin{cases} f(x), & \text{if } |x| \leq n, \\ f(2n-x), & \text{if } |x| > n. \end{cases}$$

Then f^* satisfies that $f^*(x) = f^*(2n-x)$. Although we can run Algorithm 2 to f^*, we do not know how to relate $\widetilde{\mathsf{deg}}_\epsilon(f^*)$ and $\widetilde{\mathsf{deg}}_\epsilon(f)$ for any ϵ arbitrarily close to $1/2$. Thus, the query complexity of the algorithm is not promised. We leave this case as an open problem.

5 Conclusion

This paper analyzes the quantum advantage of computing two fundamental partial symmetric Boolean functions by studying the optimal success probability of T-query quantum and randomized algorithms. Moreover, we analyze the relation between the number of queries and the bias of quantum and randomized algorithms to compute total symmetric Boolean functions when the bias of the algorithms can be arbitrarily small. Furthermore, we show the relation of several fundamental complexity measures of partial symmetric Boolean functions. We leave the fine-grained Watrous conjecture as an open problem for further study.

References

1. Scott Aaronson and Andris Ambainis. The need for structure in quantum speedups. *Theory of Computing*, 10:133–166, 2014. doi:10.4086/toc.2014.v010a006.
2. Scott Aaronson and Andris Ambainis. Forrelation: A problem that optimally separates quantum from classical computing. *SIAM Journal on Computing*, 47(3):982–1038, 2018. doi:10.1137/15M1050902.
3. Scott Aaronson and Shalev Ben-David. Sculpting quantum speedups. In *Proceedings of the 31st Conference on Computational Complexity*, volume 50, pages 26:1–26:28, 2016. doi:10.4230/LIPIcs.CCC.2016.26.
4. Scott Aaronson, Shalev Ben-David, Robin Kothari, Shravas Rao, and Avishay Tal. Degree vs. approximate degree and quantum implications of Huang's sensitivity theorem. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1330–1342. ACM, 2021. doi:10.1145/3406325.3451047.
5. Scott Aaronson, Robin Kothari, William Kretschmer, and Justin Thaler. Quantum lower bounds for approximate counting via Laurent polynomials. In *Proceedings of the 35th Computational Complexity Conference*, pages 7:1–7:47, 2020. doi:10.4230/LIPIcs.CCC.2020.7.
6. Andris Ambainis and Janis Iraids. Optimal one-shot quantum algorithm for EQUALITY and AND. *Baltic Journal of Modern Computing*, 4(4), 2016. doi:10.22364/bjmc.2016.4.4.09.
7. Arturs Backurs and Mohammad Bavarian. On the sum of L1 influences. In *Proceedings of the IEEE 29th Conference on Computational Complexity*, pages 132–143, 2014. doi:10.1109/CCC.2014.21.
8. Nikhil Bansal and Makrand Sinha. k-Forrelation optimally separates quantum and classical query complexity. In *Proccedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1303–1316, 2021. doi:10.1145/3406325.3451040.

9 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. doi:10.1145/502090.502097.

10 Shalev Ben-David. The structure of promises in quantum speedups. In *Proceedings of the 11th Conference on the Theory of Quantum Computation, Communication and Cryptography*, volume 61, pages 7:1–7:14, 2016. doi:10.4230/LIPIcs.TQC.2016.7.

11 Shalev Ben-David. Lecture 6: The polynomial method. https://cs.uwaterloo.ca/~s4bendav/CS867QIC890/CS867QIC890W21week4notes.pdf, 2021.

12 Shalev Ben-David, Andrew M. Childs, András Gilyén, William Kretschmer, Supartha Podder, and Daochen Wang. Symmetries, graph properties, and quantum speedups. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*, pages 649–660, 2020. doi:10.1109/FOCS46700.2020.00066.

13 Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. In *LATIN 1998: Theoretical Informatics, Third Latin American Symposium*, volume 1380, pages 163–169, 1998. doi:10.1007/BFb0054319.

14 Sergey Bravyi, David Gosset, Robert König, and Marco Tomamichel. Quantum advantage with noisy shallow circuits in 3D. In *Proccedings of the 60th IEEE Annual Symposium on Foundations of Computer Science*, pages 995–999, 2019. doi:10.1109/FOCS.2019.00064.

15 André Chailloux. A note on the quantum query complexity of permutation symmetric functions. In *Proceedings of the 10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 19:1–19:7, 2019. doi:10.4230/LIPIcs.ITCS.2019.19.

16 Sitan Chen, Jordan Cotler, Hsin-Yuan Huang, and Jerry Li. Exponential separations between learning with and without quantum memory. In *Proccedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science*, pages 574–585, 2021. doi:10.1109/FOCS52979.2021.00063.

17 Ronald de Wolf. A note on quantum algorithms and the minimal degree of ϵ-error polynomials for symmetric functions. *Quantum Information and Computation*, 8(10):943–950, 2008. doi:10.26421/QIC8.10-4.

18 David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 439(1907):553–558, 1992. doi:10.1098/rspa.1992.0167.

19 Bill Fefferman and Shelby Kimmel. Quantum vs. Classical Proofs and Subset Verification. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science*, volume 117, pages 22:1–22:23, 2018. doi:10.4230/LIPIcs.MFCS.2018.22.

20 Yuval Filmus, Hamed Hatami, Nathan Keller, and Noam Lifshitz. On the sum of L1 influences of bounded functions. *Israel Journal of Mathematics*, 214(1):167–192, 2016. doi:10.1007/s11856-016-1355-0.

21 Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the 18th annual ACM symposium on Theory of computing*, pages 59–68, 1986. doi:10.1145/12130.12137.

22 Daniel Grier and Luke Schaeffer. Interactive shallow clifford circuits: quantum advantage against NC^1 and beyond. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 875–888, 2020. doi:10.1145/3357713.3384332.

23 L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th IEEE Annual Symposium on Theory of Computing*, pages 212–219, 1996.

24 Buhrman Harry and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.

25 Xiaoyu He, Xiaoming Sun, Guang Yang, and Pei Yuan. Exact quantum query complexity of weight decision problems. *Science China Information Sciences*, 66:129503, 2023. Also see arXiv:1801.05717. doi:10.1007/s11432-021-3468-x.

26 Kazuo Iwama, Harumichi Nishimura, Rudy Raymond, and Shigeru Yamashita. Unbounded-error classical and quantum communication complexity. In *Proceedings of the 18th International Symposium on Algorithms and Computation, ISAAC 2007*, volume 4835, pages 100–111, 2007. doi:10.1007/978-3-540-77120-3_11.

27 Kazuo Iwama, Harumichi Nishimura, Rudy Raymond, and Shigeru Yamashita. Unbounded-error one-way classical and quantum communication complexity. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming, ICALP 2007*, volume 4596, pages 110–121, 2007. doi:10.1007/978-3-540-73420-8_12.

28 Siddharth Iyer, Anup Rao, Victor Reis, Thomas Rothvoss, and Amir Yehudayoff. Tight bounds on the Fourier growth of bounded functions on the hypercube. *arXiv preprint*, 2021. arXiv:2107.06309.

29 John Kallaugher. A quantum advantage for a natural streaming problem. In *Proccedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science*, pages 897–908, 2021. doi:10.1109/FOCS52979.2021.00091.

30 Vladimir I. Levenshtein. Krawtchouk polynomials and universal bounds for codes and designs in Hamming spaces. *IEEE Transactions on Information Theory*, 41(5):1303–1321, 1995. doi:10.1109/18.412678.

31 Shachar Lovett and Jiapeng Zhang. Fractional certificates for bounded functions. In *Proceedings of the 14th Innovations in Theoretical Computer Science Conference*, volume 251, pages 84:1–84:13, 2023. doi:10.4230/LIPIcs.ITCS.2023.84.

32 John C. Mason and David C. Handscomb. *Chebyshev polynomials*. CRC Press, 2002.

33 Ashley Montanaro, Richard Jozsa, and Graeme Mitchison. On exact quantum query complexity. *Algorithmica*, 71(4):775–796, 2015. doi:10.1007/s00453-013-9826-8.

34 Ashley Montanaro, Harumichi Nishimura, and Rudy Raymond. Unbounded-error quantum query complexity. In *Proceedings of the 19th International Symposium on Algorithms and Computation, ISAAC 2008*, volume 5369, pages 919–930, 2008. doi:10.1007/978-3-540-92182-0_80.

35 Ramamohan Paturi. On the degree of polynomials that approximate symmetric Boolean functions (preliminary version). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 468–474. ACM, 1992. doi:10.1145/129712.129758.

36 Supartha Podder, Penghui Yao, and Zekun Ye. On the fine-grained query complexity of symmetric functions. *arXiv preprint*, 2023. arXiv:2309.11279.

37 Daowen Qiu and Shenggen Zheng. Characterizations of symmetrically partial Boolean functions with exact quantum query complexity. *arXiv preprint*, 2016. arXiv:1603.06505.

38 Daowen Qiu and Shenggen Zheng. Generalized Deutsch-Jozsa problem and the optimal quantum algorithm. *Physical Review A*, 97(6):062331, 2018. doi:10.1103/PhysRevA.97.062331.

39 Daowen Qiu and Shenggen Zheng. Revisiting Deutsch-Jozsa algorithm. *Information and Computation*, 2020(275):104605, 2020. doi:10.1016/j.ic.2020.104605.

40 Alexander A. Sherstov. Approximate inclusion-exclusion for arbitrary symmetric functions. *Computational Complexity*, 18(2):219–247, 2009. doi:10.1007/s00037-009-0274-4.

41 Alexander A. Sherstov, Andrey A. Storozhenko, and Pei Wu. An optimal separation of randomized and quantum query complexity. In *Proccedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1289–1302, 2021. doi:10.1145/3406325.3451019.

42 Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. doi:10.1109/SFCS.1994.365700.

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

44 Avishay Tal. Towards optimal separations between quantum and randomized query complexities. In *Proccedings of the 61st Annual Symposium on Foundations of Computer Science*, pages 228–239, 2020. doi:10.1109/FOCS46700.2020.00030.

45 Takashi Yamakawa and Mark Zhandry. Verifiable quantum advantage without structure. In *Proceedings of the 63rd IEEE Annual Symposium on Foundations of Computer Science*, pages 69–74, 2022. doi:10.1109/FOCS54457.2022.00014.

46 Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60(4):2746, 2000. doi:10.1103/PhysRevA.60.2746.

47 Mark Zhandry. A note on the quantum collision and set equality problems. *Quantum Information and Computation*, 15(7&8):557–567, 2015. doi:10.26421/QIC15.7-8-2.

Testing Properties of Distributions in the Streaming Model

Sampriti Roy
Department of Computer Science and Engineering, IIT Madras, Chennai, India

Yadu Vasudev
Department of Computer Science and Engineering, IIT Madras, Chennai, India

Abstract

We study distribution testing in the standard access model and the conditional access model when the memory available to the testing algorithm is bounded. In both scenarios, we consider the samples appear in an online fashion. The goal is to test the properties of distribution using an optimal number of samples subject to a memory constraint on how many samples can be stored at a given time. First, we provide a trade-off between the sample complexity and the space complexity for testing identity when the samples are drawn according to the conditional access oracle. We then show that we can learn a succinct representation of a monotone distribution efficiently with a memory constraint on the number of samples that are stored that is almost optimal. We also show that the algorithm for monotone distributions can be extended to a larger class of decomposable distributions.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Property testing, distribution testing, streaming

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.56

Related Version *Full Version*: `https://arxiv.org/abs/2309.03245` [22]

1 Introduction

Sublinear algorithms that analyze massive amounts of data are crucial in many applications currently. Understanding the underlying probability distribution that generates the data is important in this regard. In the field of distribution testing, a sub-field of property testing, the goal is to test whether a given unknown distribution has a property $\mathcal{P}$ or is far from having the property $\mathcal{P}$ (where the farness is defined with respect to total variation distance). Starting from the work of Goldreich and Ron ([19]), a vast literature of work has studied the problem of testing probability distributions for important properties like identity, closeness, support size as well as properties relating to the structure of the distribution like monotonicity, k-modality, and histograms among many others; see Canonne's survey ([10]) for an overview of the problems and results.

In the works of Canonne et al ([12]) and Chakraborty et al ([13]), distribution testing with conditional samples was studied. In this model, the algorithm can choose a subset of the support, and the samples of the distribution conditioned on this subset are generated. This allows adaptive sampling from the distribution and can give better sample complexity for a number of problems. In particular, ([12]) and ([13]) give testers for uniformity and other problems that use only a constant number of samples.

The natural complexity measure of interest is the number of samples of the underlying distribution that is necessary to test the property. In many cases, when data is large, it might be infeasible to store all the samples that are generated. A recent line of work has been to study the trade-off between the sample complexity and the space complexity of algorithms

for learning and testing properties of distributions. This model can be equivalently thought of as a data stream of i.i.d samples from an unknown distribution, with the constraint that you are allowed to store only a small subset of these samples at any point in time.

In this work, we study distribution testing problems in the standard model, and when the algorithm is allowed to condition on sets to better understand the trade-off between the sample complexity and size. In particular, we study identity testing and testing whether the unknown distribution is monotone. Our work borrows ideas from the recent work of Diakonikolas et al ([17]) and extends the ideas to these problems.

1.1 Related work

Testing and estimating the properties of discrete distributions is well-studied in property testing; see ([10]) for a nice survey of recent results. In our work, we study property testing of discrete distributions under additional memory constraints wherein the algorithm does not have the resources to store all the samples that it obtains.

This line of work has received a lot of attention in recent times. Chien et al ([14]) propose a sample-space trade-off for testing any (ϵ, δ)-weakly continuous properties, as defined by Valiant ([23]). Another work by Diakonikolas et al ([17]) studies the uniformity, identity, and closeness testing problems and presents trade-offs between the sample complexity and the space complexity of the tester. They use the idea of a *bipartite collision tester* where instead of storing all the samples in the memory, the testing can be done by storing a subset of samples and counting the collisions between the stored set and the samples that come later. Another line of work ([1, 2]) focuses on the task of estimating the entropy of distributions from samples in the streaming model, where space is limited. In particular, ([1]) estimate the entropy of an unknown distribution D up to $\pm\epsilon$ using constant space. Berg et al ([7]) study the uniformity testing problem in a slightly different model where the testing algorithm is modeled as a finite-state machine.

Property testing with memory constraints has also been studied in the setting of streaming algorithms as well. Streaming algorithms were first studied in a unified way starting from the seminal work of Alon et al ([3]) where the authors studied the problem of estimating frequency moments. There is a vast amount of literature available on streaming algorithms (see [21, 20]). Bathie et al ([4]) have studied property testing in the streaming model for testing regular languages. Czumaj et al ([16]) show that every constant-query testable property on bounded-degree graphs can be tested on a random-order stream with constant space. Since this line of work is not directly relevant to our work in this paper, we will not delve deeper into it here.

1.2 Our results

In this work, we study the trade-off between sample complexity and space complexity in both the standard access model and the conditional access model. In the standard access model, a set of samples can be drawn independently from an unknown distribution. In the case of the conditional access model, a subset of the domain is given and samples can be drawn from an unknown distribution conditioned on the given set. This is similar to a streaming algorithm where the samples are presented to the algorithm, and the algorithm has a memory constraint of m bits; i.e., only up to m bits of samples can be stored in memory.

In the standard access model, which we will refer to as SAMP , we have a distribution D over the support $\{1, 2, \ldots, n\}$ and the element i is sampled with probability $D(i)$. In the conditional access model, which we will refer to as COND , the algorithm can choose a set

$S \subseteq \{1, 2, \ldots, n\}$ and will obtain samples from the conditional distribution over the set. I.e. the sample $i \in S$ is returned with probability $D(i)/D(S)$. In this work, we will work with the case when the conditioning is done on sets of size at most two - we will refer to this conditional oracle as PCOND ([12]).

Our results are stated below.

- We propose a memory-efficient identity testing algorithm in the PCOND model when the algorithm is restricted by the memory available to store the samples. We adapt the algorithm of Canonne et al ([12]) and reduce the memory requirement by using the CountMin sketch ([15]) for storing the frequencies of the samples. The identity testing algorithm uses $O(\log^2 n \log\log n/m\epsilon^2)$ samples from standard access model where $\frac{\log n\sqrt{\log\log n}}{\epsilon} \leq m \leq \frac{\log^2 n}{\epsilon}$ and an $\tilde{O}(\log^4 n/\epsilon^4)$ samples from conditional access model and does the following, if $D = D^*$, it returns Accept with probability at least 2/3, and if $d_{TV}(D, D^*) \geq \epsilon$, it returns Reject with probability at least 2/3. It uses only $O(\frac{m}{\epsilon})$ bits of memory.

  We also observe that by applying oblivious decomposition [8], performing identity and closeness testing on monotone distributions over $[n]$ can be reduced to performing the corresponding tasks on arbitrary distributions over $[O(\log(n\epsilon)/\epsilon)]$. We use the streaming model based identity tester from ([17]) and obtain an $O(\log(n\epsilon)\log\log(n\epsilon)/m\epsilon^5)$ standard access query identity tester for monotone distributions where $\log\log(n\epsilon)/\epsilon^2 \leq m \leq (\log(n\epsilon)/\epsilon)^{9/10}$. Their closeness testing algorithm also implies a closeness tester for monotone distributions which uses $O(\log(n\epsilon)\sqrt{\log\log(n\epsilon)}/\sqrt{m}\epsilon^3)$ samples from standard access model, where $\log\log(n\epsilon) \leq m \leq \tilde{\Theta}(min(\log(n\epsilon)/\epsilon, \log^{2/3}(n\epsilon)/\epsilon^2))$. Both testers require m bits of memory.
- We adapt the idea of the *bipartite collision tester* ([17]) and give an algorithm that uses $O(\frac{n\log n}{m\epsilon^8})$ samples from SAMP and tests if the distribution is monotone or far from being monotone. This algorithm requires only $O(m)$ bits of memory for $\log^2 n/\epsilon^6 \leq m \leq \sqrt{n}/\epsilon^3$. This upper bound is nearly tight since we observe that the lower bound for uniformity testing proved by Diakonikolas et al ([17]) applies to our setting as well. In particular, we show that the "no" distribution that is used in [17] is actually far from monotone, and hence the lower bound directly applies in our setting as well.
- We extend the idea of the previous algorithm for learning and testing a more general class of distribution called (γ, L)-decomposable distribution, which includes monotone and k-modal distributions. Our algorithm takes $O(\frac{nL\log(1/\epsilon)}{m\epsilon^9})$ samples from D and needs $O(m)$ bits of memory where $\log n/\epsilon^4 \leq m \leq O(\sqrt{n\log n}/\epsilon^3)$.

2 Notation and Preliminaries

Throughout this paper, we study distributions D that are supported over the set $\{1, 2, \ldots, n\} = [n]$. The notion of distance between distributions will be *total variation distance* or *statistical distance* which is defined as follows: for two distributions D_1 and D_2, the total variation distance, denoted by $d_{TV}(D_1, D_2) = \frac{1}{2}|D_1 - D_2|_1 = \frac{1}{2}\sum_{i\in[n]}|D_1(x) - D_2(x)| = max_{S\subseteq[n]}((D_1(S) - D_2(S))$. We will use $\mathcal{U}$ to denote the uniform distribution over $[n]$. We use $|.|_1$ for the ℓ_1 norm, $||.||_2$ for the ℓ_2 norm.

Let D_1 and D_2 be two distributions over $[n]$, if $d_{TV}(D_1, D_2) \leq \epsilon$, for some $0 \leq \epsilon \leq 1$, we say that D_1 is ϵ close to D_2. Let $\mathcal{D}$ be the set of all probability distributions supported on $[n]$. A property $\mathcal{P}$ is a subset of $\mathcal{D}$. We say that a distribution D is ϵ far from $\mathcal{P}$, if D is ϵ far from all the distributions having the property $\mathcal{P}$. I.e. $d_{TV}(D, D') > \epsilon$ for every $D' \in \mathcal{P}$.

We define the probability of self-collision of the distribution D by $||D||_2$. For a set S of samples drawn from D, coll(S) defines the pairwise collision count between them. Consider $S_1, S_2 \subset S$, the *bipartite collision* of D with respect to S is defined by coll(S_1, S_2) is the number of collision between S_1 and S_2.

We will be using the count of collisions among sample points to test closeness to uniformity. The following lemma connects the collision probability and the distance to uniformity.

▶ **Lemma 1** ([6]). *Let D be a distribution over $[n]$. If $\max_x D(x) \leq (1+\epsilon).\min_x D(x)$ then $||D||_2^2 \leq (1+\epsilon^2)/n$. If $||D||_2^2 \leq (1+\epsilon^2)/n$ then $d_{TV}(D,\mathcal{U}) \leq \epsilon$.*

One way to test the properties of distributions is to first learn an explicit description of the distribution. We now define the notion of flattened and reduced distributions that will be useful towards this end.

▶ **Definition 2** (Flattened and reduced distributions). *Let D be a distribution over $[n]$, and there exists a set of partitions of the domain into ℓ disjoint intervals, $\mathcal{I} = \{I_j\}_{j=1}^{\ell}$. The flattened distribution $(D^f)^{\mathcal{I}}$ corresponding to D and $\mathcal{I}$ is a distribution over $[n]$ defined as follows : for $j \in [\ell]$ and $i \in I_j$; $(D^f)^{\mathcal{I}}(i) = \frac{\sum_{t\in I_j} D(t)}{|I_j|}$. A reduced distribution D^r is defined over $[\ell]$ such that $\forall i \in \ell, D^r(i) = D(I_i)$.*

If a distribution D is ϵ close to its flattened distribution according to some partition $\{\mathcal{I}_j\}_{j=1}^{\ell}$, we refer D to be (ϵ,ℓ)-flattened. We note that if a distribution is monotonically non-increasing, then its flattened distribution is also monotonically non-increasing but its reduced distribution is not necessarily the same.

The following folklore result shows that the empirical distribution is close to the actual distribution provided sufficient number of samples are taken.

▶ **Lemma 3** (Folklore). *Given a distribution D supported over $[n]$ and an interval partition $\mathcal{I} = \{I_1, ..., I_\ell\}$, using $S = O(\frac{\ell^2}{\epsilon^2}\log\ell)$ points from SAMP, we can obtain an empirical distribution $\tilde{D}$ in the following way: $\forall I_j \in \mathcal{I}; \tilde{D}(I_j) = \frac{occ(S,I_j)}{|S|}$ ($occ(S, I_j)$ is the number of samples from S lies inside I_j) over $[\ell]$ such that for all interval I_j, with probability at least $2/3$, $|D(I_j) - \tilde{D}(I_j)| \leq \frac{\epsilon}{\ell}$. Moreover, let the flattened distribution of D be $(D^f)^{\mathcal{I}}$ and the flattened distribution of $\tilde{D}$ be $(\tilde{D}^f)^{\mathcal{I}}$, we can say that $d_{TV}((D^f)^{\mathcal{I}}, (\tilde{D}^f)^{\mathcal{I}}) < \epsilon$.*

While designing a tester for monotonicity, we use the following theorem due to Birge ([8])

▶ **Lemma 4** (Oblivious partitioning [8]). *Let D be a non-increasing distribution over $[n]$ and $\mathcal{I} = \{I_1, ..., I_\ell\}$ is an interval partitioning of D such that $|I_j| = (1+\epsilon)^j$, for $0 < \epsilon < 1$, then $\mathcal{I}$ has the following properties,*

- $\ell = O(\frac{1}{\epsilon}\log n\epsilon)$
- *The flattened distribution corresponding to $\mathcal{I}$, $(D^f)^{\mathcal{I}}$ is ϵ-close to D, or D is (ϵ,ℓ)-flattened.*

Next, we describe a data structure called the CountMin sketch which is used to estimate the frequencies of elements in a one-pass stream. It was introduced by Cormode et al ([15]). As we are dealing with a one-pass streaming algorithm with a memory constraint, it would be important to store samples in less space. CountMin sketch uses hash functions to store frequencies of the stream elements in sublinear space and returns an estimate of the same.

▶ **Definition 5** (CountMin sketch). *A CountMin (CM) sketch with parameters (ϵ,δ) is represented by a two-dimensional array counts with width w and depth d: count$[1,1], ...,$ count$[d,w]$. Set $w = \frac{e}{\epsilon}$ and $d = \log 1/\delta$. Each entry of the array is initially zero. Additionally, d hash*

functions $h_1, ..., h_d : \{1, ..., n\} \to \{1, ..., w\}$ chosen uniformly at random from a pairwise-independent family. The space requirement for the count min sketch is wd words. The sketch can be queried for the frequency of an element from the universe $\mathcal{U}$ of elements, and will return an estimate of its frequency.

The lemma below captures the fact that the frequency of any element x_i can be estimated from a CountMin sketch.

▶ **Lemma 6** ([15]). *Let $\{x_1, ..., x_S\}$ be a stream of length S and f_{x_i} be the actual frequency of an element x_i. Suppose $\tilde{f}_{x_i}$ be the stored frequency in count min sketch, then the following is true with probability at least $(1 - \delta)$, $f_{x_i} \leq \tilde{f}_{x_i} \leq f_{x_i} + \epsilon S$.*

3 Testing identity in the streaming model using PCOND

In this section, we revisit the identity testing problem using PCOND queries: given sample access and PCOND query access to an unknown distribution D we have to test whether D is identical to a fully specified distribution D^* or they are ϵ far from each other. Canonne et al ([12]) address the problem and propose a PCOND query-based identity tester. In their algorithm, the domain of D^* is divided into a set of "buckets" where the points are having almost the same weights. The algorithm samples $\tilde{O}(\log^2 n/poly(\epsilon))$ points from D and estimates the weight of each bucket. They prove if D and D^* are far then there exists at least one bucket where the weight of D^* and weight of $\tilde{D}$ will differ. If not, then the algorithm runs a process called *Compare* to estimate the ratio of the weight of each pair of points (y, z) where y is taken from a set of samples drawn from D^* and z is taken from a set of samples according to D. The following lemma is used to compare the weights of two points.

▶ **Lemma 7** ([12]). *Given as input two disjoint subsets of points X, Y together with parameters $\eta \in (0, 1], K \geq 1$ and $\delta \in (0, \frac{1}{2}]$ as well as COND query access to a distribution D, there exists a procedure* Compare *which estimates the ratio of the weights of two sets and either outputs a value $\rho > 0$ or outputs High or Low and satisfies the following:*

- *If $D(X)/K \leq D(Y) \leq K \cdot D(X)$ then with probability at least $1 - \delta$ the procedure outputs a value $\rho \in [1 - \eta, 1 + \eta]D(Y)/D(X)$;*
- *If $D(Y) > K \cdot D(X)$ then with probability at least $1 - \delta$ the procedure outputs either High or a value $\rho \in [1 - \eta, 1 + \eta]D(Y)/D(X)$;*
- *If $D(Y) < D(X)/K$ then with probability at least $1 - \delta$ the procedure outputs either Low or a value $\rho \in [1 - \eta, 1 + \eta]D(Y)/D(X)$.*

The procedure performs $O(\frac{K \log 1/\delta}{\eta^2})$ conditional queries on the set $X \cup Y$.

However, for storing $\tilde{O}(\log^2 n/poly(\epsilon))$ samples for estimating the weights of the buckets, an $\tilde{O}(\log^3 n/poly(\epsilon))$ space is required considering each sampled point takes $\log n$ bits of memory. As we are dealing with a memory constraint of m bits, for $m < O(\log^3 n)$, implementing the algorithm is not memory efficient. We use the main idea of Canonne et al ([12]), but instead of storing all samples, we use the CountMin sketch data structure for storing the frequencies of the elements of the stream. Later, the frequencies are used to estimate the weight of each bucket. By choosing the parameters of the CountMin sketch suitably, the total space required for our algorithm is at most $O(m/\epsilon)$ bits. The main concept of our algorithm lies in the theorem below,

▶ **Theorem 8** (Testing Identity [12]). *There exists an identity tester that uses an $\tilde{O}(\log^4 n/\epsilon^4)$ PCOND queries and does the following: for every pair of distributions D, D^* over $[n]$, where D^* is fully specified, the algorithm outputs Accept with probability at least $2/3$ if $D = D^*$ and outputs Reject with probability at least $2/3$ if $d_{TV}(D, D^*) \geq \epsilon$.*

Before moving into the algorithm, we define the *bucketization* technique according to ([12]). For an explicit distribution D^*, the domain is divided into ℓ buckets $\mathcal{B} = \{B_1, ..., B_\ell\}$, where each bucket contains a set of points which satisfies $B_j = \{i \in [n] : 2^{j-1}\eta/n \leq D^*(i) \leq 2^j\eta/n\}$ and $B_0 = \{i \in [n] : D^*(i) < \eta/n\}$, where $\eta = \epsilon/c$ for c to be a constant. The number of buckets $\ell = O(\lceil \log n/\eta + 1 \rceil + 1)$.

We are now ready to present our PCOND query-based one-pass streaming algorithm for identity testing. Our algorithm and the correctness borrow from ([12]) with the extra use of CountMin sketches to improve the trade-off between the sample complexity and the space used.

Algorithm 1 PCOND Identity Testing Streaming.

Input : SAMP and PCOND access to D, an explicit distribution D^*, parameters $0 < \epsilon \leq 1$, $\eta = \epsilon/6$, ℓ buckets of D^*, space requirement $O(m)$ bits $\frac{\log n\sqrt{\log\log n}}{\epsilon} \leq m \leq \frac{\log^2 n}{\epsilon}$

Output : Accept if $D = D^*$, Reject if $d_{TV}(D, D^*) \geq \epsilon$

1 Sample $S = O(\frac{\log^2 n \log\log n}{m\epsilon^2})$ points $\{x_1, ..., x_S\}$ from SAMP
2 **for** $(i = 1$ *to* $S)$ **do**
3 Estimate the frequency of x_i using CountMin sketch $(\frac{\epsilon}{m}, \frac{1}{100})$ such that $f_{x_i} \leq \tilde{f}_{x_i} \leq f_{x_i} + \frac{\epsilon}{m}S$
4 Define the frequency of each bucket B_j to be $f_{B_j} = \sum_{x_i \in B_j} f_{x_i}$, such that $f_{B_j} \leq \tilde{f}_{B_j} \leq f_{B_j} + \frac{\epsilon}{m}S^2$
5 **if** $\frac{\tilde{f}_{B_j}}{S} < D^*(B_j) - \frac{\sqrt{m}\epsilon}{\log n}$ *or* $\frac{\tilde{f}_{B_j}}{S} > D^*(B_j) + \frac{\sqrt{m}\epsilon}{\log n} + \frac{\log^2 n \log\log n}{\epsilon m^2}$ **then**
6 Reject and Exit
7 Select $s = O(\ell/\epsilon)$ points $\{y_1, ..., y_s\}$ from D^*
8 **for** *each* $y_k \in s$ **do**
9 Sample s points $\{z_1, ..., z_s\}$ from D as a stream
10 **for** *each pair of points* (y_k, z_l) *such that* $\frac{D^*(y_k)}{D^*(z_l)} \in [1/2, 2]$ **do**
11 Run *Compare* $(y_k, z_l, \eta/4\ell, 2, 1/10s^2))$
12 **if** *Compare returns Low or a value smaller than* $(1 - \eta/2\ell)\frac{D^*(y_k)}{D^*(z_l)}$ **then**
13 Reject and Exit
14 Accept

▶ **Theorem 9.** *The algorithm* PCOND IDENTITY TESTING STREAMING *uses an* $O(\log^2 n \log\log n/m\epsilon^2)$ *length stream of standard access query points and an* $\tilde{O}(\log^4 n/\epsilon^4)$ *length of conditional stream and does the following, If* $D = D^*$, *it returns* Accept *with probability at least* $2/3$, *and if* $d_{TV}(D, D^*) \geq \epsilon$, *it returns* Reject *with probability at least* $2/3$. *The memory requirement for the algorithm is* $O(\frac{m}{\epsilon})$ *(due to the parameters set in* CountMin *sketch) where* $\frac{\log n\sqrt{\log\log n}}{\epsilon} \leq m \leq \frac{\log^2 n}{\epsilon}$.

Proof.

Completeness. Suppose $D = D^*$. We prove that the algorithm does not return Reject in Line 6. Let $\tilde{D}(B_j)$ be the estimated weight of a bucket B_j where $\tilde{D}(B_j) = \frac{f_{B_j}}{S}$ for $S = O(\log^2 n \log\log n/m\epsilon^2)$. An additive Chernoff bound [followed by a union bound over the buckets] shows that with high probability, $\forall B_j, |D(B_j) - \tilde{D}(B_j)| \leq \frac{\sqrt{m}\epsilon}{\log n}$. Using Lemma 6,

with probability at least 99/100, for every element x_i in the stream, $f_{x_i} \leq \tilde{f}_{x_i} \leq f_{x_i} + \frac{\epsilon S}{m}$. Summing over all the elements in a bucket B_j, we get $\tilde{f}_{B_j} - \frac{\epsilon}{m}S^2 \leq f_{B_j} \leq \tilde{f}_{B_j}$. Substituting $\tilde{D}(B_j) = \frac{f_{B_j}}{S}$, we can see that $\frac{\tilde{f}_{B_j}}{S} - \frac{\epsilon S}{m} \leq \tilde{D}(B_j) \leq \frac{\tilde{f}_{B_j}}{S}$. As $D = D^*$, $\tilde{D}(B_j)$ is a good estimate of $D^*(B_j)$. Using $|D^*(B_j) - \tilde{D}(B_j)| \leq \frac{\sqrt{m}\epsilon}{\log n}$, we get $\frac{\tilde{f}_{B_j}}{S} - \frac{\epsilon S}{m} - \frac{\sqrt{m}\epsilon}{\log n} \leq D^*(B_j) \leq \frac{\tilde{f}_{B_j}}{S} + \frac{\sqrt{m}\epsilon}{\log n}$. This can be written as $D^*(B_j) - \frac{\sqrt{m}\epsilon}{\log n} \leq \frac{\tilde{f}_{B_j}}{S} \leq D^*(B_j) + \frac{\sqrt{m}\epsilon}{\log n} + \frac{\log^2 n \log\log n}{\epsilon m^2}$ by replacing $S = O(\log^2 n \log\log n/m\epsilon^2)$. Hence, the algorithm will not output Reject with high probability. As $D = D^*$, for all pairs (y_k, z_l) such that $\frac{D^*(y_k)}{D^*(z_l)} \in [1/2, 2]$, it follows from Lemma 7 that the estimated ratio of weights of each pair (y_k, z_l) is less than $(1 - \eta/2\ell)\frac{D^*(y_k)}{D^*(z_l)}$ [for $\eta = \epsilon/6$] with probability at most $1/10s^2$. A union bound over all $O(s^2)$ pairs proves that with a probability of at least 9/10 the algorithm outputs Accept .

Soundness. Let $d_{TV}(D, D^*) \geq \epsilon$. In this case, if one of the estimates of $\tilde{f}_{B_j}$ passes Line 5, the algorithm outputs Reject . Let's assume that the estimates are correct with high probability. The rest of the analysis follows from ([12]), we give a brief outline of the proof for making it self-contained. Define high-weight and low-weight buckets in the following way, for $\eta = \epsilon/6$, as follows: $H_j = \{x \in B_j : D(x) > D^*(x) + \eta/\ell|B_j|\}$, and $L_j = \{x \in B_j : D(x) \leq D^*(x) - \eta/\ell|B_j|\}$. It can be shown that at least one point will occur from the low-weight bucket while sampling s points in Line 7 and at least one point will come from the high-weight bucket while obtaining s points in Line 9. Using the definition of high-weight and low-weight buckets, there exists a pair (y_k, z_l) such that $D(y_k) \leq (1 - \eta/2\ell)D^*(y_k)$ and $D(z_k) > (1 + \eta/2\ell)D^*(z_k)$. By Lemma 7, with probability at least $1 - 1/10s^2$, *Compare* will return low or a value at most $(1 - \eta/2\ell)\frac{D^*(y_k)}{D^*(z_l)}$ in Line 12. Hence, the algorithm outputs Reject with high probability. ◀

We use CountMin sketch with parameters $(\frac{\epsilon}{m}, \frac{1}{100})$ in our algorithm. Comparing it with (ϵ, δ) CountMin sketch defined in ([15]), we set the width of the array to be $w = em/\epsilon$ and depth $d = \log 100$. So the space required for the algorithm is $w.d$ words which imply $O(\frac{m}{\epsilon})$ bits. For running the *Compare* procedure, we are not using any extra space for storing samples. This is because for every element in $\{y_1, ..., y_s\}$ we are sampling s length stream $\{z_1, ..., z_s\}$ and running *Compare* for each pair of points taken from each stream respectively. This leads to running compare process s^2 times. A single run of compare works in the following way in the streaming settings, for a pair (y_k, z_l), sample $O(\log^2 n/\epsilon^2)$ points from D conditioned on (y_k, z_l) and keep two counters for checking the number of times each of them appeared in the stream. Each round of *Compare* process requires $O(\log^2 n/\epsilon^2)$ length of the stream. Hence, the total stream length is $\tilde{O}(\log^4 n/\epsilon^4)$.

4 Testing Monotonicity in the streaming model using SAMP

In this section, we give an algorithm for testing monotonicity in the SAMP model when the samples are obtained via a one-pass stream. The algorithm of Batu et al ([5]) provides a sample-efficient algorithm for testing monotonicity, by dividing the support $[n]$ into intervals which are either low-weight or close to uniform. In our case, we start with the oblivious decomposition of Birge ([8]) and check if the total weight of the intervals that are far from uniform is small. To check if an interval is far from uniform, we count the number of collisions in the sample obtained from the interval. To improve the space complexity of the algorithm, we modify the part of counting collisions to counting bipartite collisions, like in ([17]). We now describe the algorithm for testing monotonicity using bipartite collisions. The sample

complexity for this algorithm is worse than the algorithm of Batu et al ([5]), but we will then show that this can be converted to an algorithm in the one-pass streaming model with better space complexity.

4.1 Testing Monotonicity using Bipartite Collisions

In this section, we perform the monotonicity testing in a slightly different fashion which functions as the building block of a streaming-based monotonicity tester. Here, unlike counting pairwise collisions between the samples, we divide the samples into two sets and count the bipartite collisions between them. The idea of the bipartite collision tester is adapted from ([17]). A key Lemma 11 proves how the bipartite collision is used to estimate the collision probability. Given sample access to an unknown distribution D over $[n]$, first, we divide the domain according to the oblivious decomposition. We count the bipartite collisions inside the intervals where enough samples lie. If D is monotone, the total weight of high collision intervals can not be too high. Prior to describing the algorithm, the lemma below clarifies the fact "enough samples" and the intervals holding them.

▶ **Lemma 10.** *Let D be a distribution over $[n]$, and $\mathcal{I} = \{I_1, ..., I_\ell\}$ be an interval partitions of $[n]$. Let $\mathcal{J} \subset \mathcal{I}$ be the set of intervals and for all $I_j \in \mathcal{J}$, $D(I_j) \geq \epsilon_1/\log n$, where $\epsilon_1 = \epsilon^2$. If $S = O(\frac{n \log n}{\epsilon^8})$ samples are drawn according to D, then all $I_j \in \mathcal{J}$ contain $|S_{I_j}| \geq O(|I_j|/\epsilon^4)$ samples.*

Proof. Fix an I_j and define a random variable, $X_i = 1$ if i^{th} sample is in I_j else 0. Let $X = \sum_{i=1}^{S} X_i = S_{I_j}$. Then the expectation $\mathbb{E}[X] = |S| \cdot D(I_j) \geq \frac{|S|\epsilon_1}{\log n}$.

By Chernoff bound, we can see that $Pr\Big[X < (1-\epsilon)\frac{|S|\epsilon_1}{\log n}\Big] = Pr\Big[X < (1-\epsilon)\mathbb{E}[X]\Big] \leq e^{-\epsilon^2 \mathbb{E}[X]} \leq e^{-\epsilon^2 \frac{|S|\epsilon^2}{\log n}} < \frac{\epsilon^2}{10 \log n}$.

The last inequality is obtained from the fact that $|S| = O(\frac{n \log n}{\epsilon^8})$ and using $\frac{n}{\epsilon^4} > \log(10 \log n/\epsilon^2)$. Applying union bound over all $\ell = O(\frac{\log n}{\epsilon_1})$ partitions, we can conclude that, $[\epsilon_1 = \epsilon^2]$ $\forall I_j$; such that $D(I_j) \geq \frac{\epsilon_1}{\log n}$ with probability at least 9/10, the following happens, $S_{I_j} \geq (1-\epsilon)\frac{|S|\epsilon_1}{\log n} \geq (1-\epsilon)\frac{n}{\epsilon^6} \geq O(|I_j|/\epsilon^4)$ ◀

The main intuition behind our algorithm is counting the bipartite collision between a set of samples. The next lemma, defines the necessary conditions for estimating collision probability using bipartite collision count.

▶ **Lemma 11.** *Let D be an unknown distribution over $[n]$ and S be the set of samples drawn according to SAMP . Let $I \subset [n]$ be an interval and S_I be the set of points lying in the interval I. Let S_I be divided into two disjoint sets S_1 and S_2; $\{S_1\} \cup \{S_2\} = \{S_I\}$ such that $|S_1| \cdot |S_2| \geq O(|S_I|/\epsilon^4)$, then with probability at least 2/3,*

$$||D_I||_2^2 - \frac{\epsilon^2}{64|I|} \leq \frac{coll(S_1, S_2)}{|S_1||S_2|} \leq ||D_I||_2^2 + \frac{\epsilon^2}{64|I|}.$$

Proof. Define the random variable $X_{ij} = 1$ if i^{th} sample in S_1 is same as j^{th} sample in S_2, 0 otherwise.

$$X = \sum_{(i,j) \in S_1 \times S_2} X_{ij} = coll(S_1, S_2)$$

$$\mathbb{E}[X] = |S_1| \cdot |S_2| \cdot ||D_I||_2^2$$

Where $||D_I||_2$ is collision probability. Let $Y_{ij} = X_{ij} - \mathbb{E}[X_{ij}] = X_{ij} - ||D_I||_2^2$.

$$\begin{aligned} Var[\sum_{(i,j)\in S_1\times S_2} X_{ij}] &= \mathbb{E}\Big[(\sum_{(i,j)\in S_1\times S_2} Y_{ij})^2\Big] \\ &= \mathbb{E}\Big[\sum_{(i,j)\in S_1\times S_2} Y_{ij}^2 + \sum_{(i,j)\neq(k,l);|\{i,j,k,l\}|=3} Y_{ij}Y_{kl}\Big] \end{aligned}$$

We calculate the following,

$$\begin{aligned} \mathbb{E}[Y_{ij}^2] &= \mathbb{E}[X_{ij}^2] - 2(\mathbb{E}[X_{ij}])^2 + (\mathbb{E}[X_{ij}])^2 \\ &= ||D_I||_2^2 - ||D_I||_2^4 \\ \mathbb{E}[Y_{ij}Y_{kl}] &= \mathbb{E}\Big[(X_{ij} - ||D_I||_2^2)(X_{kl} - ||D_I||_2^2)\Big] \\ &= \mathbb{E}\Big[X_{ij}X_{kl}\Big] - ||D_I||_2^2(\mathbb{E}[X_{ij}] + \mathbb{E}[X_{kl}]) + ||D_I||_2^4 \\ &= \mathbb{E}\Big[X_{ij}X_{kl}\Big] - ||D_I||_2^4 \end{aligned}$$

Now,

$$\begin{aligned} Var[\sum_{(i,j)\in S_1\times S_2} X_{ij}] &= \sum_{(i,j)\in S_1\times S_2} (||D_I||_2^2 - ||D_I||_2^4) + \\ &\quad \sum_{(i,j)\neq(k,l);|\{i,j,k,l\}|=3} (\mathbb{E}\Big[X_{ij}X_{kl}\Big] - ||D_I||_2^4) \\ &= |S_1|.|S_2|(||D_I||_2^2 - ||D_I||_2^4) + \sum_{(i,j);(k,j)\in S_1\times S_2; i\neq k} \mathbb{E}\Big[X_{ij}X_{kj}\Big] \\ &\quad + \sum_{(i,j);(i,l)\in S_1\times S_2; j\neq l} \mathbb{E}\Big[X_{ij}X_{il}\Big] - \sum_{(i,j)\neq(k,l);|\{i,j,k,l\}|=3} ||D_I||_2^4 \\ &= |S_1|.|S_2|(||D_I||_2^2 - ||D_I||_2^4) + |S_2|\binom{|S_1|}{2}||D_I||_3^3 \\ &\quad + |S_1|\binom{|S_2|}{2}||D_I||_3^3 - \Big(|S_2|\binom{|S_1|}{2} + |S_1|\binom{|S_2|}{2}\Big)||D_I||_2^4 \\ &\leq |S_1||S_2|\Big[(||D_I||_2^2 - ||D_I||_2^4) + (|S_1| + |S_2|)(||D_I||_3^3 - ||D_I||_2^4)\Big] \end{aligned}$$

Applying Chebyshev's inequality, we get,

$$\begin{aligned} &Pr[|X - \mathbb{E}[X]| > \frac{\epsilon^2}{64|I|}|S_1||S_2|] \leq \frac{64^2 Var[X]|I|^2}{\epsilon^4|S_1|^2|S_2|^2} \\ &\leq \frac{|S_1||S_2|\Big[(||D_I||_2^2 - ||D_I||_2^4) + (|S_1| + |S_2|)(||D_I||_3^3 - ||D_I||_2^4)\Big]64^2|I|^2}{\epsilon^4|S_1|^2|S_2|^2} \\ &\leq \frac{\Big[||D_I||_2^2 - ||D_I||_2^4 + (|S_1| + |S_2|)(||D_I||_2^3 - ||D_I||_2^4)\Big]64^2|I|^2}{\epsilon^4|S_1|.|S_2|} \\ &\leq \frac{\Big[||D_I||_2^2 - ||D_I||_2^4 + (|S_1| + |S_2|)(||D_I||_2^2 - ||D_I||_2^4)\Big]64^2|I|^2}{\epsilon^4|S_1|.|S_2|} \\ &\leq \frac{||D_I||_2^2\Big[1 - ||D_I||_2^2 + (|S_1| + |S_2|)(1 - ||D_I||_2^2)\Big]64^2|I|^2}{\epsilon^4|S_1|.|S_2|} \\ &\leq \frac{||D_I||_2^2\Big(1 - ||D_I||_2^2\Big)\Big(1 + |S_1| + |S_2|\Big)64^2|I|^2}{\epsilon^4|S_1|.|S_2|} \end{aligned}$$

Where the third inequality uses the fact that $||D_I||_3 \leq ||D_I||_2$ and the fourth inequality uses the fact that $||D_I||_2^3 \leq ||D_I||_2^2$ as $||D_I||_2 \in (0,1]$. To make the probability $< 1/3$, we have,

$$\begin{aligned}
|S_1|.|S_2| &\geq 3 \times 64^2 |I|^2 \frac{1}{\epsilon^4} ||D_I||_2^2 \Big(1 - ||D_I||_2^2\Big)\Big(1 + |S_1| + |S_2|\Big) \\
&\geq 3 \times 64^2 \frac{|I|^2}{\epsilon^4} ||D_I||_2^2 \frac{||D_I||_2^2}{100}\Big(|S_1| + |S_2|\Big) \\
&\geq 3 \times 64^2 \frac{1}{100\epsilon^4}\Big(|S_1| + |S_2|\Big) \\
&\geq O(\frac{S_I}{\epsilon^4})
\end{aligned}$$

In the second inequality we have used the fact that $(1 - ||D_I||_2^2) \geq \frac{1}{100}||D_I||_2^2$ as $||D_I||_2^2 \leq \frac{100}{101} < 1$. The third inequality is obtained from the fact that $||D||_2^2 \geq \frac{1}{|I|}$. The final inequality is obtained from the fact that $|S_I| = |S_1| + |S_2|$. Therefore, provided $|S_1|.|S_2| \geq O(\frac{|S_I|}{\epsilon^4})$, with probability at least 2/3, $||D_I||_2^2 - \frac{\epsilon^2}{64|I|} \leq \frac{coll(S_1,S_2)}{|S_1||S_2|} \leq ||D_I||_2^2 + \frac{\epsilon^2}{64|I|}$. ◀

The bipartite collision-based tester works by verifying the total weight of the intervals where the conditional distributions are far from uniformity. Let S_I be the set of samples inside an interval I and let it satisfy the condition of Lemma 11. The following lemma shows that bipartite collision count is used to detect such intervals.

▶ **Lemma 12.** *Let D be an unknown distribution over $[n]$ and $I \subset [n]$ is an interval. Let S_I be the set of points lying in the interval I and S_I can be divided into two sets S_1 and S_2 such that $|S_1||S_2| \geq O(|S_I|/\epsilon^4)$, then the following happens with probability at least 2/3*

- *If $d_{TV}(D_I, \mathcal{U}_I) > \frac{\epsilon}{4}$, then $\frac{coll(S_1,S_2)}{|S_1||S_2|} > \frac{1}{|I|} + \frac{\epsilon^2}{64|I|}$*
- *If $d_{TV}(D_I, \mathcal{U}_I) \leq \frac{\epsilon}{4}$, then, $\frac{coll(S_1,S_2)}{|S_1||S_2|} \leq \frac{1+\epsilon^2/64}{|I|} + \frac{\epsilon^2}{16}$*

Proof. Let, $d_{TV}(D_I, \mathcal{U}_I) > \frac{\epsilon}{4}$, squaring both sides, we get $(d_{TV}(D_I, \mathcal{U}_I))^2 > \frac{\epsilon^2}{16} > \frac{\epsilon^2}{32}$. Using the fact that $d_{TV}(D_I, \mathcal{U}_I) \leq \sqrt{|I|} \cdot ||D_I - \mathcal{U}_I||_2$, we deduce $|I| \cdot ||D_I - \mathcal{U}_I||_2^2 > \frac{\epsilon^2}{32}$. Simplifying the inequality, we get $||D_I - \mathcal{U}_I||_2^2 > \frac{\epsilon^2}{32|I|}$. Now, we obtain the following inequality by using $||D_I - \mathcal{U}_I||_2^2 = ||D_I||_2^2 - \frac{1}{|I|}$.

$$\begin{aligned}
||D_I||_2^2 - \frac{1}{|I|} &> \frac{\epsilon^2}{32|I|} \\
||D_I||_2^2 &> \frac{\epsilon^2}{32|I|} + \frac{1}{|I|}
\end{aligned}$$

Consider S_I is divided into two sets so that $|S_1| \cdot |S_2| \geq O(|S_I|/\epsilon^4)$, by Lemma 11 we obtain,

$$\begin{aligned}
\frac{coll(S_1,S_2)}{|S_1||S_2|} + \frac{\epsilon^2}{64|I|} &> \frac{\epsilon^2}{32|I|} + \frac{1}{|I|} \\
\frac{coll(S_1,S_2)}{|S_1||S_2|} &> \frac{1}{|I|} + \frac{\epsilon^2}{64|I|}
\end{aligned}$$

Similarly, when $d_{TV}(D_I, \mathcal{U}_I) \leq \frac{\epsilon}{4}$, we get $||D_I||_2^2 \leq \frac{\epsilon^2}{16} + \frac{1}{|I|}$. Given S_I can be divided into two sets such that $|S_1| \cdot |S_2| \geq O(|S_I|/\epsilon^4)$, by Lemma 11, $\frac{coll(S_1,S_2)}{|S_1| \cdot |S_2|} \leq \frac{1+\epsilon^2/64}{|I|} + \frac{\epsilon^2}{16}$. ◀

Now, we present the bipartite collision-based monotonicity tester.

Algorithm 2 Bipartite Collision Monotonicity.

Input : SAMP access to D, $\ell = O(\frac{1}{\epsilon_1}\log(n\epsilon_1+1))$ oblivious partitions $\mathcal{I} = \{I_1, .., I_\ell\}$ and error parameter $\epsilon, \epsilon_1 \in (0,1]$, where $\epsilon_1 = \epsilon^2$

Output : Accept if D is monotone, Reject if D is not 7ϵ close to monotone

1 Sample $T = O(\frac{1}{\epsilon^6}\log^2 n \log\log n)$ points from SAMP

2 Get the empirical distribution $\tilde{D}$ over ℓ

3 Obtain an additional sample $S = O(\frac{n\log n}{\epsilon^8})$ from SAMP

4 Let J be the set of intervals where the number of samples (in each interval I_j) is $|S_{I_j}| \geq O(|I_j|/\epsilon^4)$ and S_{I_j} can be partitioned into two disjoint sets S_1 and S_2 such that $|S_1||S_2| \geq O(|I_j|/\epsilon^8)$ and $\frac{coll(S_1,S_2)}{|S_1||S_2|} \geq (\frac{1+\epsilon^2/64}{|I_j|} + \frac{\epsilon^2}{16})$

5 **if** $\sum_{I_j\in J}\tilde{D}(I_j) > 5\epsilon$ **then**

6 Reject and Exit

7 Define a flat distribution $(\tilde{D}^f)^{\mathcal{I}}$ over $[n]$

8 Output Accept if $(\tilde{D}^f)^{\mathcal{I}}$ is 2ϵ-close to a monotone distribution. Otherwise output Reject

▶ **Theorem 13.** *The algorithm* BIPARTITE COLLISION MONOTONICITY *uses* $O(\frac{n\log n}{\epsilon^8})$ *SAMP queries and outputs Accept with probability at least* $2/3$ *if* D *is a monotone distribution and outputs Reject with probability at least* $2/3$ *when* D *is not* 7ϵ*-close to monotone.*

Proof. While sampling $O(n\log n/\epsilon^8)$ points according to D, an application of Chernoff bound shows that the intervals with $D(I_j) \geq \epsilon^2/\log n$ will contain at least $S_{I_j} = O(|I_j|/\epsilon^4)$ points. There will be at least one such interval with $D(I_j) \geq \epsilon^2/\log n$ as there are $O(\log n/\epsilon^2)$ partitions.

Completeness. Let D be monotone. By oblivious partitioning with parameter $\epsilon_1 = \epsilon^2$, we have $\sum_{j=1}^{\ell}\sum_{x\in I_j}|D(x) - \frac{D(I_j)}{|I_j|}| \leq \epsilon_1$ which implies $\sum_{j=1}^{\ell} D(I_j) d_{TV}(D_{I_j}, \mathcal{U}_{I_j}) \leq \epsilon^2$. Let J' be the set of intervals where for all I_j, $d_{TV}(D_{I_j}, \mathcal{U}_{I_j}) > \frac{\epsilon}{4}$, then $\sum_{I_j\in J'} D(I_j) \leq 4\epsilon$.

Let $\hat{J}$ is the set of intervals where $|S_1||S_2| \geq O(|S_{I_j}|/\epsilon^4)$ and $d_{TV}(D_{I_j}, \mathcal{U}_{I_j}) > \frac{\epsilon}{4}$. So, $\hat{J} \subseteq J'$. From Lemma 12, we know $\hat{J}$ is the set of intervals where $\frac{coll(S_1,S_2)}{|S_1||S_2|} > \frac{1}{|I_j|} + \frac{\epsilon^2}{64|I_j|}$. Let J be the set of intervals where $|S_1||S_2| \geq O(|S_{I_j}|/\epsilon^4)$ and $\frac{coll(S_1,S_2)}{|S_1||S_2|} > \frac{1+\epsilon^2/64}{|I_j|} + \frac{\epsilon^2}{16}$, then $J \subseteq \hat{J} \subseteq J'$. We know $\sum_{I_j\in J'} D(I_j) \leq 4\epsilon$. So, we can conclude that $\sum_{I_j\in J} D(I_j) \leq 4\epsilon$.

When $d_{TV}(D_{I_j}, \mathcal{U}_{I_j}) \leq \frac{\epsilon}{4}$, the algorithm does not sum over such $D(I_j)$ even if $|S_1||S_2| \geq O(|S_{I_j}|/\epsilon^4)$. This is because by Lemma 12 we know $\frac{coll(S_1,S_2)}{|S_1||S_2|} \leq \frac{1+\epsilon^2/64}{|I_j|} + \frac{\epsilon^2}{16}$. As a result, we can say that when D is monotone $\sum_{I_j\in J} D(I_j) \leq 4\epsilon$.

We use the empirical distribution $\tilde{D}$ and deduce that $\sum_{I_j\in J}\tilde{D}(I_j) \leq 5\epsilon$. Hence, the algorithm will NOT output Reject in Step 6. We also conclude as D is monotone, the flattened distribution $(\tilde{D}^f)^{\mathcal{I}}$ is 2ϵ close to monotone and the algorithm will output Accept in Step 8.

Soundness. We prove the contrapositive of the statement. Let the algorithm outputs Accept , then we need to prove that D is 7ϵ close to monotone.

As the algorithm accepts, $\sum_{I_j\in J}\tilde{D}(I_j) \leq 5\epsilon$, for the set of intervals J where $|S_1||S_2| \geq O(|S_{I_j}|/\epsilon^4)$ and $\frac{coll(S_1,S_2)}{|S_1||S_2|} \geq (\frac{1+\epsilon^2/64}{|I_j|} + \frac{\epsilon^2}{16})$. For all such intervals $I_j \in J$ by Lemma 11, we obtain $d_{TV}(D_{I_j}, \mathcal{U}_{I_j}) \geq \frac{\epsilon}{4}$.

Now, we calculate the distance between D and the flattened distribution and we get $d_{TV}(D, (D^f)^{\mathcal{I}}) < 4\epsilon$

We also know from Lemma 3, $d_{TV}((D^f)^{\mathcal{I}}, (\tilde{D}^f)^{\mathcal{I}}) < \epsilon$. By triangle inequality, $d_{TV}(D, (\tilde{D}^f)^{\mathcal{I}}) < 5\epsilon$. As the algorithm outputs accept, there exists a monotone distribution M, such that $d_{TV}(\tilde{D}^f)^{\mathcal{I}}, M) \leq 2\epsilon$. By triangle inequality, we have $d_{TV}(D, M) < 7\epsilon$. ◀

4.2 Testing Monotonicity in Streaming model

In this section, we present the monotonicity tester in the streaming settings. A set of samples is drawn according to the standard access model that is revealed online one at a time. The task is to test whether an unknown distribution is a monotone or ϵ far from monotonicity. Also, there is a memory bound of m bits. We use the notion of bipartite collision monotonicity tester 2 discussed in the previous section. For satisfying the memory bound, we store an optimal number of samples for such intervals and count bipartite collision between the stored samples and the remaining ones. We present the algorithm below,

Algorithm 3 Streaming Monotonicity.

Input : SAMP access to D, $\ell = O(\frac{1}{\epsilon_1}\log(n\epsilon_1 + 1))$ oblivious partitions $\mathcal{I} = \{I_1, .., I_\ell\}$ and error parameter $\epsilon, \epsilon_1 \in (0, 1]$, where $\epsilon_1 = \epsilon^2$, memory requirement $\log^2 n/\epsilon^6 \leq m \leq \sqrt{n}/\epsilon^3$

1 Sample $T = \tilde{O}(\frac{1}{\epsilon^6}\log^2 n)$ points from SAMP
2 Get the empirical distribution $\tilde{D}$ over ℓ
3 Obtain an additional sample $S = O(\frac{n\log n}{m\epsilon^8})$ from SAMP
4 For each interval store the first set of $S_1 = O(\frac{m\epsilon^2}{\log^2 n})$ samples in memory
5 Let J be the set of intervals, where for the next set of $S_2 = O(\frac{n}{m\epsilon^4})$ points, the following condition is satisfied, $\frac{coll(S_1,S_2)}{|S_1||S_2|} \geq (\frac{1+\epsilon^2/64}{|I_j|} + \frac{\epsilon^2}{16})$
6 Check **if** $\sum_{I_j \in J} \tilde{D}(I_j) > 5\epsilon$ **then**
7 ⌊ Reject and Exit
8 Define a flat distribution $(\tilde{D}^f)^{\mathcal{I}}$ over $[n]$
9 Output Accept if $(\tilde{D}^f)^{\mathcal{I}}$ is 2ϵ-close to a monotone distribution. Otherwise output Reject

▶ **Theorem 14.** *The algorithm* STREAMING MONOTONICITY *uses* $O(\frac{n\log n}{m\epsilon^8})$ *SAMP queries and outputs Accept with probability at least* $2/3$ *if* D *is a monotone distribution and outputs Reject with probability at least* $2/3$ *when* D *is not* 7ϵ *close to monotone. It uses* $O(m)$ *bits of memory for* $\log^2 n/\epsilon^6 \leq m \leq \sqrt{n}/\epsilon^3$.

Proof. As there are $O(\frac{\log n}{\epsilon^2})$ partitions, there will be at least one interval with $D(I_j) \geq \frac{\epsilon^2}{\log n}$. An application of Chernoff bound shows that with high probability all such intervals contain $|S_{I_j}| = O(n/m\epsilon^4)$ points. In the algorithm, we divide S_{I_j} into two sets S_1 and S_2 such that for $\log^2 n/\epsilon^6 \leq m \leq \sqrt{n}/\epsilon^3$, $|S_1| + |S_2| = O(m\epsilon^2/\log^2 n) + O(n/m\epsilon^4) = O(n/m\epsilon^4)$ and $|S_1|.|S_2| = O(n/\epsilon^2\log^2 n) \geq O(n/m\epsilon^8) = (1/\epsilon^4)|S_{I_j}|$. (The inequality is obtained by the fact that $m \geq \log^2 n/\epsilon^6$). This implies that the condition of Lemma 11 is satisfied by these intervals and they are eligible for estimating the collision probability using bipartite collision count. The rest of the analysis follows from Theorem 13.

The algorithm uses $O(m)$ bits of memory for implementation in a single-pass streaming model. For obtaining the empirical distribution $\tilde{D}$, we will use one counter for each of the ℓ intervals. When a sample x comes, if $x \in I_j$, the corresponding counter for I_j will be incremented by 1. In the end, the counters will give the number of samples that fall in

each of the intervals, and using those values we can explicitly obtain the distribution $\tilde{D}$. Each counter takes $O(\log n)$ bits of memory. There are total $\ell = (\log n/\epsilon^2)$ counters. So, the memory requirement for this step is $O(\log^2 n/\epsilon^2) < m$ bits. Also, using the distribution $\tilde{D}$ we can obtain the flattened distribution $(\tilde{D}^f)^{\mathcal{I}}$ without storing it explicitly. Hence, the Line 9 does not require any extra space for checking whether $(\tilde{D}^f)^{\mathcal{I}}$ is 2ϵ close to monotone or not. For storing the first set of $S_1 = O(m\epsilon^2/\log^2 n)$ samples for an interval will take $O(m\epsilon^2/\log n)$ bits of memory. As we are storing S_1 samples for all $\ell = O(\log n/\epsilon^2)$ intervals, it will take total $O(m)$ bits of memory. ◀

▶ Remark 15. If the input to the algorithm is a monotone distribution, then the streaming algorithm computes a distribution over the intervals $\mathcal{I}$ such that the flattening is close to a monotone distribution. Since the number of intervals in the partition is $O(\log n/\epsilon)$, the explicit description of the distribution can be succinctly stored.

We would also like to point out that the final step in the algorithm requires testing if the learnt distribution is close to some monotone distribution, and we have not explicitly bounded the space required for that.

4.2.1 Lower bound for testing monotonicity

In this section, we prove the lower bound for monotonicity testing problem in the streaming settings. We start with the discussion of the uniformity testing lower bound by ([17]) in the streaming model and later we show how the same lower bound is applicable in our case.

▶ **Theorem 16** (Uniformity testing lower bound in streaming framework [17]). *Let $\mathcal{A}$ be an algorithm which tests if a distribution D is uniform versus ϵ-far from uniform with error probability $1/3$, can access the samples in a single-pass streaming fashion using m bits of memory and S samples, then $S.m = \Omega(n/\epsilon^4)$. Furthermore, if $S < n^{0.9}$ and $m > S^2/n^{0.9}$ then $S \cdot m = \Omega(n \log n/\epsilon^4)$.*

The proof of the above lemma proceeds by choosing a random bit $X \in \{0, 1\}$, where $X = 0$ defines a *Yes* instance (uniform distribution) and $X = 1$ defines a *No* instance (ϵ-far from uniform) and calculating the mutual information between X and the bits stored in the memory after seeing S samples. In their formulation, the *Yes* instance is a uniform distribution over $2n$ and the *No* instance is obtained by pairing $(2i-1, 2i)$ indices together and assigning values by tossing an ϵ-biased coin. In particular, the *No* distribution is obtained as follows, pair the indices as $\{1, 2\}, \{3, 4\}, ..., \{2n-1, 2n\}$. Pick a bin $\{2i-1, 2i\}$ and for each bin a random bit $Y_i \in \{\pm 1\}$ to assign the probabilities as,

$$(D(2i-1), D(2i)) = \begin{cases} \frac{1+\epsilon}{2n}, \frac{1-\epsilon}{2n} & \text{if } Y_i = 1 \\ \frac{1-\epsilon}{2n}, \frac{1+\epsilon}{2n} & \text{if } Y_i = -1 \end{cases}$$

It is straightforward that the *Yes* distribution is a monotone distribution as well. We show that any distribution D from the *No* instance set is $O(\epsilon)$-far from monotonicity. We start by choosing an $\alpha \in (0, \epsilon/4)$ and defining a set of partitions $\mathcal{I} = \{I_1, ..., I_\ell\}$ such that $|I_j| = \lfloor (1+\alpha)^j \rfloor$ for $1 \le j \le \ell$. Let $(D^f)^{\mathcal{I}}$ be the flattened distribution corresponding to $\mathcal{I}$. We use the following lemma from ([9]) which reflects the fact if D is far from $(D^f)^{\mathcal{I}}$, then D is also far from being monotone. In particular, we define the lemma as follows,

▶ **Lemma 17** ([9]). *Let D be a distribution over domain $[n]$ and $\mathcal{I} = \{I_1, ..., I_\ell\}$ are the set of partitions defined obliviously with respect to a parameter $\alpha \in (0, 1)$ where $\ell = O(\frac{1}{\alpha} \log n\alpha)$ and $|I_j| = \lfloor (1+\alpha)^j \rfloor$. If D is ϵ-close to monotone non-increasing, then $d_{TV}(D, (D^f)^{\mathcal{I}}) \le 2\epsilon + \alpha$ where $(D^f)^{\mathcal{I}}$ is the flattened distribution of D with respect to $\mathcal{I}$.*

Let, D be a distribution chosen randomly from the *No* instance set. We have the following observation,

▶ **Lemma 18.** *Let $\mathcal{I} = \{I_1, ..., I_\ell\}$ be the oblivious partitions of D with parameter α such that $|I_j| = \lfloor (1+\alpha)^j \rfloor$.*

- *If $|I_j|$ is odd, then $\sum_{i\in I_j} |D(i) - \frac{D(I_j)}{|I_j|}| = \frac{\epsilon}{2n}(|I_j| - \frac{1}{|I_j|})$.*
- *If $|I_j|$ is even, then $\sum_{i\in I_j} |D(i) - \frac{D(I_j)}{|I_j|}| \geq \frac{\epsilon}{2n}(|I_j| - \frac{4}{|I_j|})$.*

Proof. If $|I_j|$ is odd, it will contain k (any positive integer) number of bin where each bin is of form $(2x-1, 2x)$ and an extra index i' which can have the probability weight either $\frac{1+\epsilon}{2n}$ or $\frac{1-\epsilon}{2n}$. Let $D(i') = \frac{1+\epsilon}{2n}$. In this case, $D(I_j) = \frac{|I_j|}{2n} + \frac{\epsilon}{2n}$.

$$\begin{aligned}\sum_{i\in I_j} |D(i) - \frac{D(I_j)}{|I_j|}| &= \sum_{i\in I_j} |D(i) - \frac{1}{2n} - \frac{\epsilon}{2n|I_j|}| \\ &= \frac{\epsilon}{2n}(1 - \frac{1}{|I_j|})\frac{|I_j|-1}{2} + \frac{\epsilon}{2n}(1 + \frac{1}{|I_j|})\frac{|I_j|-1}{2} + \frac{\epsilon}{2n}(1 - \frac{1}{|I_j|}) \\ &= \frac{\epsilon}{2n}(|I_j| - \frac{1}{|I_j|})\end{aligned}$$

When $D(i') = \frac{1-\epsilon}{2n}$, similar calculation will follow.

If $|I_j|$ is even, there are two possibilities, (i) I_j consists of k (positive integer) bins. So, there will be equal number of $\frac{1+\epsilon}{2n}$ and $\frac{1-\epsilon}{2n}$ in I_j and $D(I_j) = \frac{|I_j|}{2n}$. In this case, it is straightforward to observe that $\sum_{i\in I_j} |D(i) - \frac{D(I_j)}{|I_j|}| = \frac{\epsilon|I_j|}{2n}$. Another case is, (ii) I_j contains $b_p, ..., b_{p+k-1}$ bins completely and $i' \in b_{p-1}$, and $i'' \in b_{p+k}$ where $D(i') = D(i'')$; the case when $D(i') \neq D(i'')$ will be similar to (i) that we saw earlier. Let $D(i') = D(i'') = \frac{1+\epsilon}{2n}$. In this case, $D(I_j) = \frac{|I_j|}{2n} + \frac{\epsilon}{n}$.

$$\begin{aligned}\sum_{i\in I_j} |D(i) - \frac{D(I_j)}{|I_j|}| &= \sum_{i\in I_j} |D(i) - \frac{1}{2n} - \frac{\epsilon}{n|I_j|}| \\ &= \frac{\epsilon}{2n}(1 - \frac{1}{|I_j|})\frac{|I_j|-2}{2} + \frac{\epsilon}{2n}(1 + \frac{1}{|I_j|})\frac{|I_j|-2}{2} + \frac{\epsilon}{n}(1 - \frac{2}{|I_j|}) \\ &= \frac{\epsilon}{2n}(|I_j| - \frac{4}{|I_j|})\end{aligned}$$

Combining (i) and (ii), we say $\sum_{i\in I_j} |D(i) - \frac{D(I_j)}{|I_j|}| \geq \frac{\epsilon}{2n}(|I_j| - \frac{4}{|I_j|})$. Similar calculation will follow when $D(i') = D(i'') = \frac{1-\epsilon}{2n}$. ◀

In our case, we apply oblivious partitions on D (chosen randomly from the *No* set) with respect to the parameter α and use the above lemma, to conclude the following,

▶ **Lemma 19.** *Let D be a randomly chosen distribution from the No instance set, then D is $\epsilon/4$-far from any monotone non-increasing distribution.*

Proof. We calculate $d_{TV}(D, (D^f)^{\mathcal{I}}) = \sum_{j=1}^{\ell} \sum_{i\in I_j} |D(i) - \frac{D(I_j)}{|I_j|}| = \sum_{|I_j| \text{is even}} \sum_{i\in I_j} |D(i) - \frac{D(I_j)}{|I_j|}| + \sum_{|I_j| \text{is odd}} \sum_{i\in I_j} |D(i) - \frac{D(I_j)}{|I_j|}|$. Each odd length interval contributes $\sum_{i\in I_j} |D(i) - \frac{D(I_j)}{|I_j|}| = \frac{\epsilon}{2n}(|I_j| - \frac{1}{|I_j|})$ and each even length interval contributes $\sum_{i\in I_j} |D(i) - \frac{D(I_j)}{|I_j|}| \geq \frac{\epsilon}{2n}(|I_j| - \frac{4}{|I_j|})$.

Hence, simplifying the distance, we get, $d_{TV}(D,(D^f)^{\mathcal{I}}) \geq \sum_{|I_j| \text{is even}} \frac{\epsilon}{2n}(|I_j| - \frac{4}{|I_j|}) + \sum_{|I_j| \text{is odd}} \frac{\epsilon}{2n}(|I_j| - \frac{1}{|I_j|}) \geq \frac{\epsilon}{2n}\sum_{I_j \in \ell}|I_j| - \frac{\epsilon}{2n}\big(\sum_{|I_j| \text{is even}} \frac{4}{|I_j|} + \sum_{|I_j| \text{is odd}} \frac{1}{|I_j|}\big) \geq \epsilon - \frac{\epsilon}{2n}.5\ell \geq \frac{3\epsilon}{4} > 2\frac{\epsilon}{4} + \alpha$, for $\alpha = \epsilon/4$. The third inequality is obtained by using the fact that $|I_j| \geq 1$ and the fourth inequality considers $\ell < n/10$. Now, by using the contra-positive of the Lemma 17, D is $\epsilon/4$-far from any monotone non-increasing distribution. ◀

Therefore, the uniformity testing lower bound from [17] is applicable in our case for distinguishing monotone from $\epsilon/4$-far monotone. We formalize this in the theorem below.

▶ **Theorem 20.** *Let $\mathcal{A}$ be an algorithm that tests if a distribution D is monotone versus $\epsilon/4$-far from monotonicity with error probability $1/3$, can access the samples in a single-pass streaming fashion using m bits of memory and S samples, then $S.m = \Omega(n/\epsilon^4)$. Furthermore, if $n^{0.34}/\epsilon^{8/3} + n^{0.1}/\epsilon^4 \leq m \leq \sqrt{n}/\epsilon^3$, then $S.m = \Omega(n \log n/\epsilon^4)$.*

We obtain the above theorem as analogous to the Theorem 16 by showing that lower bound for uniformity implies lower bound for monotonicity in the streaming framework. In particular, the uniform distribution is monotone non-increasing by default and we show that a randomly chosen distribution from *No* instance set is $\epsilon/4$-far from monotone no-increasing. Hence, the correctness of the above theorem follows directly from the Theorem 16.

4.3 Learning decomposable distributions in the streaming model

The algorithm and analysis from the previous section of monotone distributions extend to a more general class of structured distributions known as (γ, L)-decomposable distributions ([11, 18]). Formally, the class of (γ, L)-decomposable distributions is defined as follows.

▶ **Definition 21** ((γ, L)-decomposable distribution). *A class $\mathcal{C}$ of distributions is said to be (γ, L)-decomposable, if for every $D \in \mathcal{C}$, there exists an $\ell \leq L$ and a partition $\mathcal{I} = \{I_1, .., I_\ell\}$ of $[n]$ into intervals such that for every interval $I_j \in \mathcal{I}$ one of the following conditions hold.*

- $D(I_j) \leq \frac{\gamma}{L}$
- $max_{i \in I_j} D(i) \leq (1+\gamma) min_{i \in I_j} D(i)$

In particular, monotone distributions, k-modal distributions, k-histograms are (γ, L)-decomposable for suitable values of γ and L. We refer to the appendix for a discussion regarding the same. We can use the ideas from the previous section and modify the algorithm of Fischer et al ([18]) to obtain trade-offs between the sample complexity and space complexity for learning the class of (γ, L)-decomposable distributions. In particular, we have the following theorem,

▶ **Theorem 22.** *If D is an $(\epsilon/2000, L)$-decomposable distribution, then the algorithm* LEARNING L-DECOMPOSABLE DISTRIBUTION STREAMING *outputs a distribution $(\tilde{D}^f)^{\mathcal{I}}$ such that $d_{TV}(D,(\tilde{D}^f)^{\mathcal{I}}) \leq \epsilon$ with probability at least $1-\delta$. The algorithm requires $O(\frac{nL\log(1/\epsilon)}{m\epsilon^9})$ samples from D and needs $O(m)$ bits of memory where $\log n/\epsilon^4 \leq m \leq O(\sqrt{n\log n}/\epsilon^3)$.*

5 Conclusion

We give efficient algorithms for testing identity, monotonicity and (γ, L)-decomposability in the streaming model. For a memory constraint m, the number of samples required is a function of the support size n and the constraint m. For monotonicity testing, our bounds are nearly optimal. We note that the trade-off that we achieve, and lower bounds work for

certain parameters of the value m. Furthermore, we have not tried to tighten the dependence of the bound on the parameter ϵ. One natural question to ask is if the dependence of sample complexity on m can be improved, and whether it can work for a larger range of values.

References

1 Jayadev Acharya, Sourbh Bhadane, Piotr Indyk, and Ziteng Sun. Estimating entropy of distributions in constant space. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

2 Maryam Aliakbarpour, Andrew McGregor, Jelani Nelson, and Erik Waingarten. Estimation of entropy in constant space with improved sample complexity. *arXiv preprint arXiv:2205.09804*, 2022.

3 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

4 Gabriel Bathie and Tatiana Starikovskaya. Property testing of regular languages with applications to streaming property testing of visibly pushdown languages. In *ICALP 2021*, GLASGOW (virtual conference), United Kingdom, 2021.

5 T. Batu, L. Fortnow, E. Fischer, R. Kumar, R. Rubinfeld, and P. White. Testing random variables for independence and identity. In *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 442–451, Washington, DC, USA, 2001. IEEE Computer Society.

6 Tugkan Batu, Ravi Kumar, and Ronitt Rubinfeld. Sublinear algorithms for testing monotone and unimodal distributions. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 381–390, New York, NY, USA, 2004. ACM.

7 Tomer Berg, Or Ordentlich, and Ofer Shayevitz. On the memory complexity of uniformity testing. In Po-Ling Loh and Maxim Raginsky, editors, *Proceedings of Thirty Fifth Conference on Learning Theory*, volume 178 of *Proceedings of Machine Learning Research*, pages 3506–3523. PMLR, 02–05 July 2022.

8 Lucien Birge. On the Risk of Histograms for Estimating Decreasing Densities. *The Annals of Statistics*, 15(3):1013–1022, 1987.

9 Clément L. Canonne. Big Data on the rise: Testing monotonicity of distributions. In *42nd International Conference on Automata, Languages and Programming (ICALP)*, 2015.

10 Clément L. Canonne. Topics and techniques in distribution testing: A biased but representative sample. *Found. Trends Commun. Inf. Theory*, 19(6):1032–1198, 2022. doi:10.1561/0100000114.

11 Clément L. Canonne, Ilias Diakonikolas, Themis Gouleakis, and Ronitt Rubinfeld. Testing shape restrictions of discrete distributions. *Theory of Computing Systems*, 62(1):4–62, January 2018. Publisher Copyright: © 2017, Springer Science+Business Media New York.

12 Clément L. Canonne, Dana Ron, and Rocco A. Servedio. Testing probability distributions using conditional samples. *SIAM Journal on Computing*, 44(3):540–616, 2015.

13 Sourav Chakraborty, Eldar Fischer, Yonatan Goldhirsh, and Arie Matsliah. On the power of conditional samples in distribution testing. *SIAM Journal on Computing*, 45(4):1261–1296, 2016.

14 Steve Chien, Katrina Ligett, and Andrew McGregor. Space-efficient estimation of robust statistics and distribution testing. In Andrew Chi-Chih Yao, editor, *Innovations in Computer Science – ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 251–265. Tsinghua University Press, 2010.

15 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. In *J. Algorithms*, 2004.

16 Artur Czumaj, Hendrik Fichtenberger, Pan Peng, and Christian Sohler. Testable properties in general graphs and random order streaming. In *24th International Conference on Randomization and Computation (RANDOM)*, 2020.

17 Ilias Diakonikolas, Themis Gouleakis, Daniel M. Kane, and Sankeerth Rao. Communication and memory efficient testing of discrete distributions. In *Annual Conference Computational Learning Theory*, 2019.

18 Eldar Fischer, Oded Lachish, and Yadu Vasudev. Improving and extending the testing of distributions for shape-restricted properties. *Algorithmica, Springer*, 81,3765–3802, 2019. `arXiv:1609.06736`.

19 Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. *Electron Colloq Comput Complexity*, 7, January 2000.

20 Andrew McGregor. Graph stream algorithms: A survey. *SIGMOD Rec.*, 43(1):9–20, May 2014.

21 Shanmugavelayutham Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

22 Sampriti Roy and Yadu Vasudev. Testing properties of distributions in the streaming model, 2023. `arXiv:2309.03245`.

23 Paul Valiant. Testing symmetric properties of distributions. *SIAM Journal on Computing*, 40(6):1927–1968, 2011.

A Strongly Polynomial-Time Algorithm for Weighted General Factors with Three Feasible Degrees

Shuai Shao
School of Computer Science and Technology & Hefei National Laboratory,
University of Science and Technology of China, Hefei, China.

Stanislav Živný
Department of Computer Science, University of Oxford, UK

Abstract

General factors are a generalization of matchings. Given a graph G with a set $\pi(v)$ of feasible degrees, called a degree constraint, for each vertex v of G, the general factor problem is to find a (spanning) subgraph F of G such that $\deg_F(v) \in \pi(v)$ for every v of G. When all degree constraints are symmetric Δ-matroids, the problem is solvable in polynomial time. The weighted general factor problem is to find a general factor of the maximum total weight in an edge-weighted graph. Strongly polynomial-time algorithms are only known for weighted general factor problems that are reducible to the weighted matching problem by gadget constructions.

In this paper, we present a strongly polynomial-time algorithm for a type of weighted general factor problems with real-valued edge weights that is provably not reducible to the weighted matching problem by gadget constructions. As an application, we obtain a strongly polynomial-time algorithm for the terminal backup problem by reducing it to the weighted general factor problem.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases matchings, factors, edge constraint satisfaction problems, terminal backup problem, delta matroids

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.57

Related Version *Full Version*: https://arxiv.org/abs/2301.11761

Funding The research leading to these results has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). This research was also funded by UKRI EP/X024431/1. All data is provided in full in the results section of this paper. Part of the work was done while the first author was a postdoctoral research associate at the University of Oxford.

1 Introduction

A matching in an undirected graph is a subset of the edges that have no vertices in common, and it is perfect if its edges cover all vertices of the graph. Graph matching is one of the most studied problems both in graph theory and combinatorial optimization, with beautiful structural results and efficient algorithms described, e.g., in the monograph of Lovász and Plummer [38] and in relevant chapters of standard textbooks [43, 34]. In particular, the *weighted (perfect) matching problem* is to find a (perfect) matching of the maximum total weight for a given graph of which each edge is assigned a weight. This problem can be solved in polynomial time by the celebrated Edmonds' blossom algorithm [14, 15]. Since then, a number of more efficient algorithms have been developed [20, 35, 31, 8, 22, 27, 24, 23, 26, 29]. Table III of [10] gives a detailed review of these algorithms.

The *f-factor problem* is a generalization of the perfect matching problem in which one is given a non-negative integer $f(v)$ for each vertex $v \in V$ of $G = (V, E)$. The task is to find a (spanning) subgraph $F = (V_F, E_F)$ of G such that $\deg_F(v) = f(v)$ for every $v \in V$.[1] The case $f(v) = 1$ for every $v \in V$ is the perfect matching problem. This problem, as well as the weighted version, can be solved efficiently by a gadget reduction to the perfect matching problem [16]. In addition, Tutte gave a characterization of graphs having an f-factor [47], which generalizes his characterization theorem for perfect matchings [46]. Subsequently, the study of graph factors has attracted much attention with many variants of graph factors, e.g., b-matchings, $[a, b]$-factors, (g, f)-factors, parity (g, f)-factors, and anti-factors introduced, and various types of characterization theorems proved for the existence of such factors. We refer the reader to the book [1] and the survey [40] for a comprehensive treatment of the developments on the topic of graph factors.

In the early 1970s, Lovász introduced a generalization of the above factor problems [36, 37], for which we will need a few definitions. For any nonnegative integer n, let $[n]$ denote $\{0, 1, \ldots, n\}$. A degree constraint D of arity n is a subset of $[n]$.[2] We say that a degree constraint D has a gap of length k if there exists $p \in D$ such that $p + 1, \ldots, p + k \notin D$ and $p + k + 1 \in D$. An instance of the *general factor problem* (GFP) [36, 37] is given by a graph $G = (V, E)$ and a mapping π that maps every vertex $v \in V$ to a degree constraint $\pi(v) \subseteq [\deg_G(v)]$ of arity $\deg_G(v)$. The task is to find a subgraph, if one exists, F of G such that $\deg_F(v) \in \pi(v)$ for every $v \in V$. The case $\pi(v) = \{0, 1\}$ for every $v \in V$ is the matching problem, and the case $\pi(v) = \{1\}$ for every $v \in V$ is the perfect matching problem. Cornuéjols showed that the GFP is solvable in polynomial time if each degree constraint has gaps of length at most 1 [7]. When a degree constraint having a gap of length at least 2 occurs, the GFP is NP-complete [37, 7] except for the case when all constraints are either 0-*valid* or 1-*valid*. A degree constraint D of arity k is 0-valid if $0 \in D$, and 1-valid if $k \in D$. When all constraints are 0-*valid*, the empty graph is a factor. When all constraints are 1-*valid*, the underlying graph is a factor of itself. In both cases, the GFP is trivially tractable.

In this paper, we consider the *weighted general factor problem* (WGFP) where each edge is assigned a real-valued weight and the task is to find a general factor of the maximum total weight. We suppose that each degree constraint has gaps of length at most 1 for which the unweighted GFP is known to be polynomial-time solvable. Some cases of the WGFP are reducible to the weighted matching or perfect matching problem by gadget constructions, and hence are polynomial-time solvable. In these cases, the degree constraints are called *matching-realizable* (see Definition 18). For instance, the degree constraint $D = [b]$ where $b > 0$, for b-matchings is matching realizable [48]. The weighted b-matching problem is interesting in its own right in combinatorial optimization and has been well studied with many elaborate algorithms developed [41, 39, 21, 3, 25]. Besides b-matchings, Cornuéjols showed that the *parity interval* constraint $D = \{g, g + 2, \ldots, f\}$ where $f \geq g \geq 0$ and $f \equiv g \mod 2$, is matching realizable [7], and Szabó showed that the *interval* constraint $D = \{g, g + 1, \ldots, f\}$ where $f \geq g \geq 0$, for (g, f)-factors is matching realizable [45]. Thus, the WGFP where each degree constraint is an interval or a parity interval is reducible to the weighted matching problem (with some vertices required to have degree exactly 1) and hence solvable in polynomial-time by Edmonds' algorithm, although Szabó gave a different

[1] In graph theory, a graph factor is usually a spanning subgraph. Here, without causing ambiguity, we allow F to be an arbitrary subgraph including the empty graph and we adapt the convention that $\deg_F(v) = 0$ if $v \in V \setminus V_F$.

[2] We always associate a degree constraint with an arity. Two degree constraints are different if they have different arities although they may be the same set of integers.

algorithm for this problem [45]. By reducing the WGFP with interval and parity interval constraints to the weighted (g, f)-factor problem, a faster algorithm was obtained in [11] based on Gabow's algorithm [21].

In [45], Szabó further conjectured that the WGFP is solvable in polynomial time without requiring each degree constraint should be an interval or a parity interval, as long as each degree constraint has gaps of length at most 1. To prove the conjecture, a natural question is then the following: *Are there other WGFPs that are polynomial-time solvable by a gadget reduction to weighted matchings?* In other words, *are there other degree constraints that are matching realizable?* In this paper, we show that the answer is *no*.

▶ **Theorem 1.** *A degree constraint with gaps of length at most* 1 *is matching realizable if and only if it is an interval or a parity interval.*

Previous results beyond matchings realizable degree constraints. With the answer to the above question being negative, new algorithms need to be devised for the WGFP with degree constraints that are not intervals or parity intervals. Unlike the weighted matching problem and the weighted b-matching problem for which various types of algorithms have been developed, only a few algorithms have been presented for the more general and challenging WGFP: For the cardinality version of WGFP, i.e., the WGFP where each edge is assigned weight 1, Dudycz and Paluch introduced a polynomial-time algorithm for this problem with degree constrains having gaps of length at most 1, which leads to a pseudo-polynomial-time algorithm for the WGFP with non-negative integral edge weights [11].

The algorithm in [11] is based on a structural result showing that if a factor is not optimal, then a factor of larger weight can be found by a local search, which can be done in polynomial time. However, it is not clear how much larger the weight of the new factor is. In order to get an optimal factor, the algorithm needs to repeat local searches iteratively until no better factors can be found, and the number of local searches is bounded by the total edge weight, which makes the algorithm pseudo-polynomial-time. Later, in an updated version [12], by carefully assigning edge weights, the algorithm was improved to be weakly polynomial-time with a running time $O(\log W m n^6)$, where W is the largest edge weight, m is the number of edges and n is the number of vertices. Later, Kobayash extended the algorithm to a more general setting called jump system intersections [33].

Our main contribution. Independently of [12], in this paper, we make a step towards a *strongly* polynomial-time algorithm for the WGPF. Let $p \geq 0$ be an arbitrary integer. Consider the following two types of degree constraints $\{p, p+1, p+3\}$ and $\{p, p+2, p+3\}$ (of arbitrary arity). We will call them *type-1* and *type-2* respectively. These are the "smallest" degree constraints that are not matching realizable.

▶ **Theorem 2.** *There is a strongly polynomial-time algorithm for the* WGFP *with real-valued edge weights where each degree constraint is an interval, a parity interval, a type-1, or a type-2 (of arbitrary arities). The algorithm runs in time $O(n^6)$ for a graph with n vertices.*

The requirement of degree constraints in our result may look overly specific. However, the scope of our algorithm is not narrow. First, our result implies a complexity dichotomy for the WGFP on all subcubic graphs (see the following Theorem 3), which for many is a large and interesting class of graphs. More importantly, there are interesting problems arising from applied areas that are encompassed by the WGPF with constraints considered in this paper.

For instance, the *terminal backup problem* from network design is the following problem. Given a graph consisting of terminal nodes, non-terminal nodes, and edges with non-negative costs. The goal is to find a subgraph with the minimum total cost such that each terminal node

is connected to at least one other terminal node (for the purpose of backup in applications). It is known that an optimal solution of the terminal backup problem consists of edge-disjoint paths containing 2 terminals and stars containing 3 terminals [49]. In other words, in an optimal subgraph of the terminal backup problem, each terminal node has degree 1 and each non-terminal node has degree 0, 2 or 3. Thus, the terminal backup problem can be expressed as a WGFP with degree constraints $\{1\}$ and $\{0, 2, 3\}$ (both of arbitrary arities). A weakly polynomial-time algorithm was given for the terminal backup problem in [2]. Our result gives a strongly polynomial-time algorithm for this problem.

In addition, our algorithm gives a tractability result for the WGFP with degree constraints that are provably not matching realizable, thus going beyond existing algorithms. The algorithm is a recursive algorithm, reducing the problem to a smaller sub-problem of itself by fixing the parity of degree constraints on vertices. Its correctness is based on a delicate structural result, which is stronger than that of [12].[3] Equipped with this result, our algorithm can directly find an *optimal* factor (not just a better one) of an instance of a larger size by performing only one local search from an optimal factor of a smaller instance. Here, the important part is not how to find a better factor by local search (the main result of [12]) but rather how to ensure that the better factor obtained by only one local search is actually optimal under certain assumptions. This is the key to making our algorithm strongly polynomial. In addition, as a by-product, we give a simple proof of the result of [12] for the special case of WGFP with interval, parity interval, type-1 and type-2 degree constraints by reducing the problem to WGFP on subcubic graphs and utilizing the equivalence between 2-vertex connectivity and 2-edge connectivity of subcubic graphs.

Relation with (edge) constraint satisfaction problems. The graph factor problem is encompassed by a special case of the Boolean constraint satisfaction problem (CSP), called edge-CSP, in which every variable appears in exactly two constraints [30, 17]. When every constraint is symmetric (i.e, the value of the constraint only depends on the Hamming weight of its input), the Boolean edge-CSP is a graph factor problem.

For general Boolean edge-CSPs, Feder showed that the problem is NP-complete if a constraint that is not a Δ-matroid occurs, except for those that are tractable by Schaefer's dichotomy theorem for Boolean CSPs [42]. In a subsequent line of work [9, 28, 18, 13], tractability of Boolean edge-CSPs has been established for special classes of Δ-matroids, most recently for even Δ-matroids [32]. A complete complexity classification of Boolean edge-CSPs is still open with the conjecture that all Δ-matroids are tractable. A degree constraint (i.e., a symmetric constraint) is a Δ-matroid if and only if it has gaps of length at most 1. Thus, the above conjecture holds for symmetric Boolean edge-CSPs by Cornuéjols' result on the general factor problem [7]. A complexity classification for weighted Boolean edge-CSPs is certainly a more challenging goal: The complexity of weighted Boolean edge-CSPs with even Δ-matroids as constraints is still open. Our result in Theorem 2 gives a tractability result for weighted Boolean edge-CSPs with certain symmetric Δ-matroids as constraints. Combining our main result with known results on Boolean valued CSPs [6], we obtain a complexity dichotomy for weighted Boolean edge-CSPs with symmetric constraints of arity no more than 3, i.e., the WGFP on subcubic graphs.

[3] The result in [12] holds for the more general WGFP with all degree constraints having gaps of length at most 1, while our result only works for the WGFP with interval, parity interval, type-1 and type-2 degree constraints.

Let D be a degree constraint of arity at most 3. If $D \neq \{0,3\}$, then D is an interval, a parity interval, a type-1, or a type-2. Thus, if the constraint $\{0,3\}$ (of arity 3) does not occur, then the WGFP is strongly polynomial-time solvable by our main theorem. Otherwise, the constraint $\{0,3\}$ occurs. In this case, for a vertex v labeled by $\{0,3\}$, the three edges incident to it must take the same assignments in a feasible factor (i.e., the three edges are all either present or absent in any factor). Thus, the vertex v can be viewed as a Boolean variable and it appears in three other degree constraints connected to it. By viewing all vertices with $\{0,3\}$ as variables appearing three times and the other edges as variables appearing twice, the WGFP becomes a special case of valued CSPs where some variables appear three times and the other variables appear twice. It is known that once variables are allowed to appear three times in a CSP, then they can appear arbitrarily many times [9]. Thus, the WGPF with $\{0,3\}$ occurring is equivalent to a standard (non-edge) CSP [19]. By the dichotomy theorem for valued CSPs [6], one can check that the problem is tractable if and only if for every degree constraint D of arity $k \leq 3$, $D \subseteq \{0,k\}$. Thus, we have the following complexity dichotomy.

▶ **Theorem 3.** *The* WGFP *on subcubic graphs is strongly polynomial-time solvable if*
1. *the degree constraint* $\{0,3\}$ *of arity* 3 *does not occur,*
2. *or for every degree constraint* D *of arity* $k \leq 3$, $D \subseteq \{0,k\}$.

Otherwise, the problem is NP-hard.

Organization. In Section 2, we present basic definitions and notation. In Section 3, we describe our algorithm and give a structural result for the WGFP that ensures the correctness and the strongly polynomial-time running time of our algorithm. In Section 4, we introduce basic augmenting subgraphs as an analogy of augmenting paths for weighed matchings and give a proof of the structural result. The proof is based on a result regarding the existence of certain basic factors for subcubic graphs, for which we give a proof sketch in Section 5. Finally, we discuss matching realizability and its relation with Δ-matroids in Appendix A. All omitted proofs can be found in the full version [44].

2 Preliminaries

Let $\mathcal{D}$ be a (possibly infinite) set of degree constraints.

▶ **Definition 4.** *The* weighted general factor problem *parameterized by* $\mathcal{D}$*, denoted by* WGFP($\mathcal{D}$)*, is the following computational problem. An instance is a triple* $\Omega = (G, \pi, \omega)$*, where* $G = (V, E)$ *is a graph,* $\pi : V \to \mathcal{D}$ *assigns to every* $v \in V$ *a degree constraint* $D_v \in \mathcal{D}$ *of arity* $\deg_G(v)$*, and* $\omega : E \to \mathbb{R}$ *assigns to every* $e \in E$ *a real-valued weight* $w(e) \in \mathbb{R}$*. The task is to find, if one exists, a general factor* F *of* G *such that the total weight of edges in* F *is maximized.*

The general factor problem GFP($\mathcal{D}$) *is the decision version of* WGFP($\mathcal{D}$)*; i.e., deciding whether a general factor exists or not.*

Suppose that $\Omega = (G, \pi, \omega)$ is a WGFP instance. If F is a general factor of G under π, then we say that F is a factor of Ω, denoted by $F \in \Omega$. In terms of this inclusion relation, Ω can be viewed as a set of subgraphs of G. We extend the edge weight function ω to subgraphs of G. For a subgraph H of G, its weight $\omega(H)$ is $\sum_{e \in E(H)} \omega(e)$ ($\omega(H) = 0$ if H is the empty graph). If H contains an isolated vertex v, then $\omega(H) = \omega(H')$, where H' is the graph obtained from H by removing v. Moreover, $H \in \Omega$ if and only if $H' \in \Omega$. In the

following, without other specification, we always assume that a factor does not contain any isolated vertices. The optimal value of Ω, denoted by $\mathrm{Opt}(\Omega)$, is $\max_{F\in\Omega}\omega(F)$. We define $\mathrm{Opt}(\Omega)=-\infty$ if Ω has no factor. A factor F of Ω is *optimal* in Ω if $\omega(F)=\mathrm{Opt}(\Omega)$.

For a WGFP instance $\Omega'=(G',\pi',\omega')$, where $G'\subseteq G$[4] and ω' is the restriction of ω on the edges of G', we say Ω' is a *sub-instance* of Ω, denoted by $\Omega'\subseteq\Omega$, if $F\in\Omega$ for every $F\in\Omega'$. In particular, Ω' is a subset of Ω by viewing them as two sets of subgraphs of G. If $\Omega'\subseteq\Omega$, then $\mathrm{Opt}(\Omega')\le\mathrm{Opt}(\Omega)$. For two WGFP instances $\Omega_1=(G,\pi_1,\omega)$ and $\Omega_2=(G,\pi_2,\omega)$, we use $\Omega_1\cup\Omega_2$ to denote the union of factors of these two instances, i.e., $\Omega_1\cup\Omega_2=\{F\subseteq G\mid F\in\Omega_1 \text{ or } F\in\Omega_2\}$, and $\Omega_1\cap\Omega_2$ to denote the intersection, i.e., $\Omega_1\cap\Omega_2=\{F\subseteq G\mid F\in\Omega_1 \text{ and } F\in\Omega_2\}$. Note that $\Omega_1\cup\Omega_2$ and $\Omega_1\cap\Omega_2$ are sets of subgraphs of G and may not define WGFP instances on G.

We use $\mathcal{G}_1$ and $\mathcal{G}_2$ to denote the set of degree constraints that are intervals and parity intervals, respectively, and $\mathcal{T}_1$ and $\mathcal{T}_2$ to denote the set of degree constraints that are type-1 and type-2, respectively. Let $\mathcal{G}=\mathcal{G}_1\cup\mathcal{G}_2$ and $\mathcal{T}=\mathcal{T}_1\cup\mathcal{T}_2$. In this paper, we study the problem $\mathrm{WGFP}(\mathcal{G}\cup\mathcal{T})$.

Let $H_1=(V_1,E_1)$ and $H_2=(V_2,E_2)$ be two subgraphs of G. The symmetric difference graph $H_1\Delta H_2$ is the induced subgraph of G induced by the edge set $E_1\Delta E_2$. Note that there are no isolated vertices in a symmetric difference graph. When $E_1\cap E_2=\emptyset$, we may write $H_1\Delta H_2$ as $H_1\cup H_2$. When $E_2\subseteq E_1$, we may write $H_1\Delta H_2$ as $H_1\backslash H_2$. A *subcubic* graph is defined to be a graph where every vertex has degree $1,2$ or 3. Unless stated otherwise, we use V_G and E_G to denote the vertex set and the edge set of a graph G, respectively.

3 Algorithm

We give a recursive algorithm for the problem $\mathrm{WGFP}(\mathcal{G}\cup\mathcal{T})$, using the problems $\mathrm{WGFP}(\mathcal{G})$ and the decision problem $\mathrm{GFP}(\mathcal{G}\cup\mathcal{T})$ as oracles.

Given an instance $\Omega=(G,\pi,\omega)$ of $\mathrm{WGFP}(\mathcal{G}\cup\mathcal{T})$, we define the following sub-instances of $\Omega=(G,\pi,\omega)$ that will be used in the recursion. Recall that V_G denotes the vertex set of the graph G. Let T_Ω denote the set $\{v\in V_G\mid\pi(v)\in\mathcal{T}\}$. (We may omit the subscript Ω of T_Ω when it is clear from the context.)

For every vertex $v\in T_\Omega$, we split the instance Ω in two by splitting the degree constraint $\pi(v)$ in two parity intervals. More precisely, we define

$$\begin{aligned} &D_v^0=\{p_v+1,p_v+3\} \text{ and } D_v^1=\{p_v\} && \text{if} \quad \pi(v)=\{p_v,p_v+1,p_v+3\}\in\mathcal{T}_1;\\ &D_v^0=\{p_v,p_v+2\} \text{ and } D_v^1=\{p_v+3\} && \text{if} \quad \pi(v)=\{p_v,p_v+2,p_v+3\}\in\mathcal{T}_2. \end{aligned}$$

We have $D_v^0,D_v^1\in\mathcal{G}_2$. For $i\in\{0,1\}$ and $v\in T_\Omega$, we define $\Omega_v^i=(G,\pi_v^i,\omega)$ to be the sub-instance of Ω where $\pi_v^i(x)=\pi(x)$ for every $x\in V_G\backslash\{v\}$ and $\pi_v^i(v)=D_v^i$. Then, for every $v\in T_\Omega$, we have $\Omega_v^0\cap\Omega_v^1=\emptyset$ and $\Omega_v^0\cup\Omega_v^1=\Omega$. Moreover, $T_{\Omega_v^0}=T_{\Omega_v^1}=T_\Omega\backslash\{v\}$.

Let F be a factor of Ω. Similarly to above, one can partition Ω into $2^{|T_\Omega|}$ many sub-instances according to F such that each one is an instance of $\mathrm{WGFP}(\mathcal{G})$ – for each $v\in T_\Omega$, we choose one of the two splits of $\pi(v)$ as above. (We note that the algorithm will not consider all exponentially many sub-instances.) In detail, for every vertex $v\in T_\Omega$, we define $D_v^F=D_v^i$ where $\deg_F(v)\in D_v^i$ as follows:

[4] We use the term "subgraph" and notation $G'\subseteq G$ throughout for the standard meaning of a "normal" subgraph i.e., if $G=(V',E')$ and $G=(V,E)$ then $G'\subseteq G$ means $V'\subseteq V$ and $E'\subseteq E$.

$$
\begin{aligned}
D_v^F &= \{p_v\} && \text{if} \quad \pi(v) = \{p_v, p_v+1, p_v+3\} \in \mathfrak{T}_1 \quad \text{and} \quad \deg_F(v) = p_v, \\
D_v^F &= \{p_v+1, p_v+3\} && \text{if} \quad \pi(v) = \{p_v, p_v+1, p_v+3\} \in \mathfrak{T}_1 \quad \text{and} \quad \deg_F(v) \neq p_v; \\
D_v^F &= \{p_v+3\} && \text{if} \quad \pi(v) = \{p_v, p_v+2, p_v+3\} \in \mathfrak{T}_2 \quad \text{and} \quad \deg_F(v) = p_v+3, \\
D_v^F &= \{p_v, p_v+2\} && \text{if} \quad \pi(v) = \{p_v, p_v+2, p_v+3\} \in \mathfrak{T}_2 \quad \text{and} \quad \deg_F(v) \neq p_v+3.
\end{aligned}
$$

By definition, $\deg_F(v) \in D_v^F \subseteq \pi(v)$ and $D_v^F \in \mathcal{G}_2$. In fact, D_v^F is the maximal set such that $\deg_F(v) \in D_v^F \subseteq \pi(v)$ and $D_v^F \in \mathcal{G}_2$. One can also check that for every $v \in T_\Omega$, $\pi(v) \backslash D_v^F \in \mathcal{G}_2$, and moreover for every $p \in D_v^F$ and $q \in \pi(v) \backslash D_v^F$, $p \not\equiv q \mod 2$.

For every $W \subseteq T_\Omega$, we define $\Omega_W^F = (G, \pi_W^F, \omega)$ to be the sub-instance of Ω where $\pi_W^F(v) = \pi(v) \backslash D_v^F$ for $v \in W$, $\pi_W^F(v) = D_v^F$ for $v \in T_\Omega \backslash W$, and $\pi_W^F(v) = \pi(v)$ for $v \in V \backslash T_\Omega$. Then for every W, Ω_W^F is an instance of WGFP($\mathcal{G}$). Moreover, we have $\cup_{W \subseteq T} \Omega_W^F = \Omega$ and $\Omega_{W_1}^F \cap \Omega_{W_2}^F = \emptyset$ for every $W_1 \neq W_2$. Thus, $\{\Omega_W^F\}_{W \subseteq T_\Omega}$ is a partition of Ω (viewed as a set of subgraphs of G). When $W = \emptyset$, we write Ω_W^F as Ω^F.

Our algorithm is given in Algorithm 1.

Algorithm 1 Finding an optimal factor for an instance of WGFP($\mathcal{G} \cup \mathfrak{T}$).

Function `Decision`**:**
  Input : An instance $\Omega = (G, \pi, \omega)$ of WGFP($\mathcal{G} \cup \mathfrak{T}$).
  Output : A factor of Ω, or "No" if Ω has no factor.

Function `Optimization`**:**
  Input : An instance $\Omega = (G, \pi, \omega)$ of WGFP($\mathcal{G}$).
  Output : An optimal factor of Ω, or "No" if Ω has no factor.

Function `Main`**:**
  Input : An instance $\Omega = (G, \pi, \omega)$ of WGFP($\mathcal{G} \cup \mathfrak{T}$).
  Output : An optimal factor $F \in \Omega$, or "No" if Ω has no factor.

```
T ← {v ∈ V | π(v) ∈ 𝔗};
if T is the empty set then
    return Optimization (Ω);
else
    Arbitrarily pick u ∈ T;
    if Decision (Ω_u^0) returns "No" then
        return Main (Ω_u^1);
    else
        F^opt ← Main (Ω_u^0);
        foreach v ∈ T do
            // Elements of T can be traversed in an arbitrary order.
            W ← {u} ∪ {v};
            if Optimization(Ω_W^{F^opt}) ≠ "No" then F' ← Optimization(Ω_W^{F^opt});
            if ω(F') > ω(F^opt) then F^opt ← F';
        end
        return F^opt;
    end
end
```

The key that makes our algorithm running strongly polynomial-time is the following structural result (Theorem 5) for the problem WGFP($\mathcal{G} \cup \mathcal{T}$). It says that given an optimal factor F of Ω_u^0 for some $u \in T_\Omega$, if F is not optimal in Ω, then we can directly find an optimal factor of Ω by searching at most n sub-instances of Ω which are in WGFP($\mathcal{G}$). Note that the number of searches is independent of the edge weights. Thus, the problem of finding an optimal factor in Ω can be reduced to finding an optimal factor in Ω_u^0, where there is one fewer vertex u with constraints in $\mathcal{T}$. By recursively reducing an instance to another with fewer vertices with constraints in $\mathcal{T}$, we eventually get an instance of WGFP($\mathcal{G}$) which can be solved in polynomial-time. This leads to a strongly polynomial-time algorithm for finding an optimal factor.

▶ **Theorem 5.** *Suppose that $\Omega = (G, \pi, \omega)$ is an instance of* WGFP($\mathcal{G} \cup \mathcal{T}$)*, F is a factor of Ω and F is optimal in Ω_u^0 for some $u \in T_\Omega$. Then a factor F' is optimal in Ω if and only if $\omega(F') \geq \omega(F)$ and $\omega(F') \geq \mathrm{Opt}(\Omega_W^F)$ for every W where $u \in W \subseteq T_\Omega$ and $|W| = 1$ or 2.*

In other words, if F is not optimal in Ω, then there is an optimal factor of Ω which belongs to Ω_W^F for some W where $u \in W \subseteq T_\Omega$ and $|W| = 1$ or $|W| = 2$.

▶ Remark 6. This result is *stronger* than the main result (Theorem 2) of [12], and it is *not* simply implied by [12]. To clarify this, we give a simple proof outline of Theorem 5 here.

In order to prove Theorem 5, it suffices to prove the direction that if $\omega(F') \geq \omega(F)$ and $\omega(F') \geq \mathrm{Opt}(\Omega_W^F)$ for every W where $u \in W \subseteq T_\Omega$ and $|W| = 1$ or 2, then F' is optimal in Ω. We prove this by contradiction. Suppose that F' is not optimal in Ω, and F^* is an optimal factor of Ω. Then, $\omega(F^*) > \omega(F') \geq \mathrm{Opt}(\Omega_W^F)$ for every $W \subseteq T_\Omega$ where $|W| \leq 2$. Also, $\omega(F^*) \notin \Omega_u^0$ since $\omega(F^*) > \omega(F) = \mathrm{Opt}(\Omega_u^0)$. Thus, $\deg_{F^*}(u) \not\equiv \deg_F(u) \mod 2$.

By [12], a *canonical path* $M \subseteq F \Delta F^*$ with positive weight[5] can be found, and then $F \Delta M$ is a factor of Ω with larger weight than F and $F \Delta M \in \Omega_W^F$ for some $W \subseteq T_\Omega$ where $|W| \leq 2$. However, this does not lead to a contradiction. To get a contradiction, we need to show that the positive weighted canonical path M (a basic augmenting subgraph) further satisfies $\deg_M(u) \equiv 0 \mod 2$. Then, $\deg_{F \Delta M}(u) \equiv \deg_F(u) \mod 2$. Thus, $F \Delta M$ is a factor with larger weight than F and $F \Delta M \in \Omega_u^0$, which contradicts with F being optimal in Ω_u^0.

The existence of a basic augmenting subgraph M satisfying $\deg_M(u) \equiv 0 \mod 2$ is formally stated in the second property of Lemma 12. The main technical part of the paper (Section 5.2 of the full paper) is devoted to prove it. In Section 5 of this short version, we give an example to illustrate the proof ideas. The existence of such a basic augmenting subgraph is highly non-trivial. In fact, it does *not* hold anymore after a subtle change of the condition "F is optimal in $\Omega_u^{\mathbf{0}}$" to "F is optimal in $\Omega_u^{\mathbf{1}}$" for some $u \in T_\Omega$. We give the following example (see Figure 1) to show this.

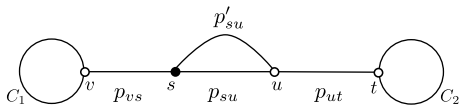


Figure 1 An example that violates Theorem 5 when F is optimal in Ω_u^1 instead of Ω_u^0.

In this instance, $\pi(u) = \pi(v) = \pi(t) = \{0, 1, 3\}$ (denoted by hollow nodes) and $\pi(s) = \{0, 2, 3\}$ (denoted by the solid node), and $\omega(C_1) = \omega(p_{vs}) = \omega(p_{su}) = \omega(p'_{su}) = \omega(p_{ut}) = \omega(C_2) = 1$. Inside the cycles C_1 and C_2, and the paths p_{vs}, p_{su}, p_{ut}, and p'_{su}, there are

[5] See definition 3 of [12]. They are defined as basic augmenting subgraphs (Definition 11) in this paper.

other vertices of degree 2 with the degree constraint $\{0, 2\}$ so that the graph G is simple. We omit these vertices of degree 2 in Figure 1. In this case, $T_\Omega = \{u, v, s, t\}$. Consider the sub-instance $\Omega^1_u = (G, \pi^1_u, \omega)$. We have $\pi^1_u(u) = D^1_u = \{0\}$ since $\pi(u) = \{0, 1, 3\}$. Then, the only factor F of Ω^1_u is the empty graph (assuming there are no isolated vertices in factors), and F is not optimal in Ω. Also, the only optimal factor of Ω is the graph G and $G \in \Omega^F_{T_\Omega}$ where $|T_\Omega| = 4$. Clearly, $\deg_G(u) \not\equiv \deg_F(u) \mod 2$. One can check that for any factor F' of Ω with larger weight than F, $\deg_{F'}(u) \not\equiv \deg_F(u) \mod 2$. In other words, there is no basic augmenting subgraph M such that $\deg_M(u) \equiv 0 \mod 2$. Moreover, one can check that in this case, Theorem 5 also does not hold. In other words, the existence of a basic augmenting subgraph satisfying $\deg_M(u) \equiv 0 \mod 2$ is crucial for the correctness of Theorem 5.

Using Theorem 5, we now prove that Algorithm 1 is correct.

▶ **Lemma 7.** *Given an instance $\Omega = (G, \pi, \omega)$ of* WGFP$(\mathcal{G}, \mathcal{T})$*, Algorithm 1 returns either an optimal factor of Ω, or "No" if Ω has no factor.*

Proof. Recall that for an instance $\Omega = (G, \pi, \omega)$, we define $T_\Omega = \{v \in V_G \mid \pi(v) \in \mathcal{T}\}$ where V_G is the vertex set of G . We prove the correctness by induction on the $|T_\Omega|$.

If $|T_\Omega| = 0$, Ω is an instance of WGFP$(\mathcal{G})$. Algorithm 1 simply returns `Optimization` (Ω). By the definition of the function `Optimization`, the output is correct.

Suppose that Algorithm 1 returns correct results for all instances Ω' of WGFP$(\mathcal{G}, \mathcal{T})$ where $|T_{\Omega'}| = k$. We consider an instance Ω of WGFP$(\mathcal{G}, \mathcal{T})$ where $|T_\Omega| = k + 1$. Algorithm 1 first calls the function `Decision` (Ω^0_u) for some arbitrary $u \in T$.

We first consider the case that `Decision` (Ω^0_u) returns "No". By the definition, Ω^0_u has no factor. Moreover, since $\Omega = \Omega^0_u \cup \Omega^1_u$, we have $F \in \Omega$ if and only if $F \in \Omega^1_u$. Then, a factor $F \in \Omega^1_u$ is optimal in Ω if and only if it is optimal in Ω^1_u. Note that Ω^1_u is an instance of WGFP$(\mathcal{G}, \mathcal{T})$ where $|T_{\Omega^1_u}| = k$. By the induction hypothesis, Algorithm 1 returns a correct result `Main` (Ω^1_u) for the instance Ω^1_u, which is also a correct result for the instance Ω.

Now, we consider the case that `Decision` (Ω^0_u) returns a factor of Ω^0_u. Then, `Main` (Ω^0_u) returns an optimal factor F of Ω^0_u. After the loop (lines 13 to 17) in Algorithm 1, we get a factor F^{opt} of Ω such that $\omega(F^{\text{opt}}) \geq \text{Opt}(\Omega^F_W)$ for every $u \in W \subseteq T_\Omega$ where $|W| = 1$ (when $u = v$) or $|W| = 2$ (when $u \neq v$) and $\omega(F^{\text{opt}}) \geq \omega(F)$. By Theorem 5, F^{opt} is an optimal factor of Ω. Thus, Algorithm 1 returns a correct result. ◀

Now, we consider the time complexity of Algorithm 1. The size of an instance is defined to be the number of vertices of the underlying graph of the instance.

▶ **Lemma 8.** *Run Algorithm 1 on an instance $\Omega = (G, \pi, \omega)$ of size n. Then,*

- *the algorithm will stop the recursion after at most n recursive steps;*
- *the algorithm will call* `Decision` *at most n many times, call* `Optimization` *at most $\frac{n(n+1)}{2} + 1$ many times, and perform at most $\frac{n(n+1)}{2}$ many comparisons;*
- *the algorithm runs in time $O(n^6)$.*

Proof. Let $\Omega^k = (G, \pi^k, \omega)$ be the instance after k many recursive steps. Here $\Omega^0 = \Omega$. Recall that $T_{\Omega^k} = \{v \in V \mid \pi^k(v) \in \mathcal{T}\}$. For an instance Ω^k with $|T_{\Omega^k}| > 0$, the recursive step will then go to the instance $(\Omega^k)^0_u$ or $(\Omega^k)^1_u$ for some $u \in T_{\Omega^k}$. Thus, $\Omega^{k+1} = (\Omega^k)^0_u$ or $(\Omega^k)^1_u$. In both cases, $T_{\Omega^{k+1}} = T_{\Omega^k} \backslash \{u\}$ and hence $|T_{\Omega^{k+1}}| = |T_{\Omega^k}| - 1$. By design, the algorithm will stop the recursion and return `Optimization` (Ω^m) when it reaches an instance Ω^m with $|T_{\Omega^m}| = 0$. Thus, #recursive steps $= m = |T_\Omega| - 0 \leq |V| = n$. To prove the second item, we consider the number of operations inside the recursive step for the instance $\Omega^k = (G, \pi^k, \omega)$. Note that $k \leq n$ and $|T_{\Omega^k}| = |T_\Omega| - k \leq n - k$. If $|T_{\Omega^k}| = 0$, then the algorithm will simply call `Optimization`

once. If $|T_{\Omega^k}| > 0$, then inside the recursive step, the algorithm will call `Decision` once, and call `Optimization` once or $|T_{\Omega^k}|$ many times depending on the answer of `Decision`. Moreover, in the later case, the algorithm will also perform $|T_{\Omega^k}|$ many comparisons. Thus, we have #calls of `Decision` $= \sum_{|T_{\Omega^k}|>0} 1 = \sum_{i=1}^{|T_\Omega|} 1 = |T_\Omega| \leq n$, #calls of `Optimization` $\leq 1+\sum_{|T_{\Omega^k}|>0} |T_{\Omega^k}| = 1+\sum_{i=1}^{|T_\Omega|} i \leq \frac{n(n+1)}{2}+1$, and #comparisons $\leq \sum_{|T_{\Omega^k}|>0} |T_{\Omega^k}| \leq \frac{n(n+1)}{2}$. Let $t_{\texttt{Main}}(n)$ denote the running time of Algorithm 1 on an instance of size n, and $t_{\texttt{Dec}}(n)$ and $t_{\texttt{Opt}}(n)$ denote the running time of algorithms for functions `Decision` and `Optimization`, respectively. Then, $t_{\texttt{Dec}}(n) = O(n^4)$ by the algorithm in [7] and $t_{\texttt{Opt}}(n) = O(n^4)$ by the algorithm in [11]. Thus, $t_{\texttt{Main}}(n) \leq nt_{\texttt{Dec}}(n) + \frac{n(n+1)+2}{2} t_{\texttt{Opt}}(n) + \frac{n(n+1)}{2} = O(n^6)$. ◀

4 Proof of Theorem 5

In this section, we give a proof of Theorem 5. The general strategy is that starting with a non-optimal factor F of an instance $\Omega = (G, \omega, \pi)$, we want to find a subgraph H of G such that by taking the symmetric difference $F\Delta H$, we get another factor of Ω with larger weight. The existence of such subgraphs is trivial (Lemma 10). However, the challenge is how to find one efficiently. As an analogy of augmenting paths in the weighted matching problem, we introduce basic augmenting subgraphs (Definition 11) for the weighted graph factor problem, which can be found efficiently. We will show that given a non-optimal factor F, a basic augmenting subgraph always exists (Lemma 12, property 1). Then, we can efficiently improve the factor F to another factor with larger weight. As shown in [12], this already gave a weakly polynomial-time algorithm. However, the existence of basic augmenting subgraphs is not enough to get a strongly polynomial-time algorithm, which requires the number of improvement steps being independent of edge weights. Thus, in order to prove Theorem 5, which leads to a strongly polynomial-time algorithm, we further establish that there exists a basic augmenting subgraph that satisfies certain stronger properties under suitable assumptions (Lemma 12, property 2). This result will imply Theorem 5.

▶ **Definition 9** (F-augmenting subgraphs). *Suppose that F is a factor of an instance $\Omega = (G, \pi, \omega)$. A subgraph H of G is F-augmenting if $F\Delta H \in \Omega$ and $\omega(F\Delta H) - \omega(F) > 0$.*

▶ **Lemma 10.** *Suppose that F is a factor of an instance Ω. If F is not optimal in Ω, then there exists an F-augmenting subgraph.*

Proof. Since F is not optimal, there is some $F' \in \Omega$ such that $\omega(F') > \omega(F)$. Let $H = F\Delta F'$. We have $F\Delta H = F' \in \Omega$ and $\omega(H) = \omega(F') - \omega(F) > 0$. Thus, H is F-augmenting. ◀

Recall that for an instance $\Omega = (G, \pi, \omega)$ of WGFP$(\mathcal{G}, \mathcal{T})$, T_Ω is the set $\{v \in V_G \mid \pi(v) \in \mathcal{T}\}$. For two factors $F, F^* \in \Omega$, we define $T_\Omega^{F\Delta F^*} = \{v \in T_\Omega \mid \deg_{F\Delta F^*}(v) \equiv 1 \mod 2\}$.

▶ **Definition 11** (Basic augmenting subgraphs). *Suppose that F and F^* are factors of an instance $\Omega = (G, \pi, \omega)$ and $\omega(F) < \omega(F^*)$. An F-augmenting subgraph $H = (V_H, E_H)$ is (F, F^*)-*basic *if $H \subseteq F\Delta F^*$, $|V_H^{\text{odd}}| \leq 2$, and $V_H^{\text{odd}} \cap T_\Omega \subseteq T_\Omega^{F\Delta F^*}$ where $V_H^{\text{odd}} = \{v \in V_H \mid \deg_H(v) \equiv 1 \mod 2\}$.*

▶ **Lemma 12.** *Suppose that F and F^* are two factors of an instance $\Omega = (G, \pi, \omega)$.*

1. *If $\omega(F^*) > \omega(F)$, then there exists an (F, F^*)-basic subgraph.*
2. *If $\omega(F^*) > \text{Opt}(\Omega_W^F)$ for every $W \subseteq T_\Omega^{F\Delta F^*}$ with $|W| \leq 2$, and $T_\Omega^{F\Delta F^*}$ contains a vertex u such that $F \in \Omega_u^0$ (i.e., $\deg_F(u) \in D_u^0$), then there exists an (F, F^*)-basic subgraph H where $\deg_H(u) \equiv 0 \mod 2$.*

▶ Remark 13. The first property of Lemma 12 implies the following: a factor $F \in \Omega$ is optimal if and only if $\omega(F) \geq \text{Opt}(\Omega_W^F)$ for every $W \subseteq T_\Omega$ with $|W| \leq 2$. This is a special case of the main result (Theorem 2) of [12] where the authors consider the WGFP for all constraints with gaps of length at most 1. The second property of Lemma 12 is more refined than the first property and it implies our main result (Theorem 5). In this paper, as a by-product of the proof of property 2, we give a simple proof of Theorem 2 of [12] for the special case WGFP($\mathcal{G} \cup \mathcal{T}$) based on certain properties of subcubic graphs.

Using the second property of Lemma 12, we can prove Theorem 5.

▶ **Theorem** (Theorem 5). *Suppose that F is a factor of an instance $\Omega = (G, \pi, \omega)$, and F is optimal in Ω_u^0 for some $u \in T_\Omega$. Then a factor F' is optimal in Ω if and only if $\omega(F') \geq \omega(F)$ and $\omega(F') \geq \text{Opt}(\Omega_W^F)$ for every W where $u \in W \subseteq T_\Omega$ and $|W| = 1$ or 2.*

Proof. If F' is optimal in Ω, then clearly $\omega(F') \geq \omega(F)$ and $\omega(F') \geq \text{Opt}(\Omega_W^F)$ for every W where $u \in W \subseteq T_\Omega$ and $|W| = 1$ or 2. Thus, to prove the theorem, it suffices to prove the other direction. Since $\omega(F') \geq \omega(F)$ and F is optimal in Ω_u^0, we have $\omega(F') \geq \text{Opt}(\Omega_W^F)$ for every $W \subseteq T_\Omega$ where $u \notin W$ and $|W| \leq 2$. Also, since $\omega(F') \geq \text{Opt}(\Omega_W^F)$ for every W where $u \in W \subseteq T_\Omega$ and $|W| = 1$ or 2, we have $\omega(F') \geq \text{Opt}(\Omega_W^F)$ for every $W \subseteq T_\Omega$ where $|W| \leq 2$. For a contradiction, suppose that F' is not optimal in Ω. Let F^* be an optimal factor of Ω. Then, $\omega(F^*) > \omega(F')$. Thus, $\omega(F^*) > \omega(F') \geq \text{Opt}(\Omega_W^F)$ for every $W \subseteq T_\Omega$ where $|W| \leq 2$. Also, $F^* \notin \Omega_u^0$ since $\omega(F^*) > \omega(F)$ and F is optimal in Ω_u^0. Thus, $\deg_{F^*}(u) \not\equiv \deg_F(u) \mod 2$. Then, $T_\Omega^{F\Delta F^*}$ contains the vertex u such that $F \in \Omega_u^0$. By Lemma 12, there exists an (F, F^*)-basic subgraph H where $\deg_H(u) \equiv 0 \mod 2$. Let $F'' = F\Delta H$. Then $F'' \in \Omega$ and $\omega(F'') > \omega(F)$. Also, $F'' \in \Omega_u^0$ since $\deg_{F''}(u) \equiv \deg_F(u) \mod 2$. This is a contradiction with F being optimal in Ω_u^0. ◀

Now it suffices to prove Lemma 12. By a type of normalization maneuver, we can transfer any instance of WGFP($\mathcal{G}, \mathcal{T}$) to an instance of WGFP($\mathcal{G}, \mathcal{T}$) defined on subcubic graphs, called a key instance (Definition 14). Recall that a subcubic graph is a graph where every vertex has degree 1, 2 or 3. For key instances, there are five possible forms of basic augmenting subgraphs, called basic factors (Definition 15). Then, the crux of the proof of Lemma 12 is to establish the existence of certain basic factors of key instances (Theorem 16). For a proof of Lemma 12 using Theorem 16, please refer to the proof of Lemma 4.4 in the full paper.

▶ **Definition 14** (Key instance). *A* key instance $\Omega = (G, \pi, \omega)$ *is an instance of* WGFP($\mathcal{G}, \mathcal{T}$) *where G is a subcubic graph, and for every $v \in V_G$, $\pi(v) = \{0, 1\}$ if $\deg_G(v) = 1$, $\pi(v) = \{0, 2\}$ if $\deg_G(v) = 2$, and $\pi(v) = \{0, 1, 3\}$ (i.e., type-1) or $\{0, 2, 3\}$ (i.e., type-2) if $\deg_G(v) = 3$. We say a vertex v of degree 3 is of type-1 or type-2 if $\pi(v)$ is type-1 or type-2 respectively. We say a vertex v of any degree is 1-feasible or 2-feasible if $1 \in \pi(v)$ or $2 \in \pi(v)$ respectively.*

▶ **Definition 15** (Basic factor). *Let Ω be a key instance. A factor of Ω is a* basic factor *if it is in one of the following five forms: a* path*, a* cycle*, a* tadpole graph *(i.e., a graph consisting of a cycle and a path such that they intersect at one endpoint of the path), a* dumbbell graph *(i.e., a graph consisting of two vertex disjoint cycles and a path such that the path intersects with each cycle at one of its endpoints), and a* theta graph *(i.e., a graph consisting of three vertex disjoint paths with the same two endpoints).*

▶ **Theorem 16.** *Suppose that $\Omega = (G, \pi, \omega)$ is a key instance.*

1. *If $\omega(G) > 0$, then there is a basic factor F of Ω such that $\omega(F) > 0$.*
2. *If $\omega(G) > 0$, $\omega(G) > \omega(F)$ for every basic factor F of Ω, and G contains a vertex u with $\deg_G(u) = 1$ or $\deg_G(u) = 3$ and $\pi(u) = \{0, 2, 3\}$, then there is a basic factor F^* of Ω such that $\omega(F^*) > 0$ and $\deg_{F^*}(u) \equiv 0 \mod 2$. (Recall that $\deg_{F^*}(u) = 0$ if $u \notin V_{F^*}$.)*

▶ Remark 17. For the second property of Theorem 16, the requirement of $\pi(u) = \{0, 2, 3\}$ when $\deg_G(u) = 3$ is crucial. Consider the instance $\Omega = (G, \pi, \omega)$ as shown in Figure 1. Note that Ω is a key instance. and $\pi(u) = \{0, 1, 3\}$. In this case where $\pi(u) = \{0, 1, 3\}$, it can be checked that the second property does *not* hold.

5 Proof Sketch of Theorem 16

In this section, we give a proof sketch of Theorem 16 and we focus on the proof of the second property using the first property. Omitted proofs can be found in Section 5 of the full paper.

Proof sketch. By property 1 of Theorem 16, there exists at least one basic factor of Ω such that its weight is positive. Among all such basic factors, we pick an F such that $\omega(F)$ is the largest. Consider the graph $G' = G \backslash F$, i.e., the subgraph of G induced by the edge set $E_G \backslash E_F$. We consider the instance $\Omega' = (G', \pi', \omega')$ where for every $x \in V_{G'}$, $\pi'(x) = \{0, 1\}$ if $\deg_{G'}(x) = 1$, $\pi'(x) = \{0, 2\}$ if $\deg_{G'}(x) = 2$ and $\pi'(x) = \pi(x)$ if $\deg_{G'}(x) = 3$, and ω' is the weight function ω restricted to G'. Note that Ω' is also a key instance, but it is not necessarily a sub-instance of Ω. Since $\omega(G) > \omega(F)$, we have $\omega'(G') = \omega(G') = \omega(G) - \omega(F) > 0$. Without causing ambiguity, we may simply write ω' as ω in the instance Ω'. By property 1 of Theorem 16, there exists a basic factor F' of Ω' such that $\omega(F') > 0$. Since $E_{F'} \subseteq E_G \backslash E_F$, F and F' are edge-disjoint. Let $H = F \cup F'$, which is the subgraph of G induced by the edge set $E_F \cup E_{F'}$. We will show that we can find a subgraph F^* of H such that F^* is the desired basic factor of Ω satisfying $\omega(F^*) > 0$ and $\deg_{F^*}(u) \equiv 0 \mod 2$.

First, we show that H is a factor of Ω. Let $V_\cap = V_F \cap V_{F'}$. We show that for every $x \in V_H \backslash V_\cap$, $\deg_H(x) \in \pi(x)$. If $x \in V_F \backslash V_\cap$, then $\deg_H(x) = \deg_F(x)$. Since $F \in \Omega$, $\deg_F(x) \in \pi(x)$. Then, $\deg_H(x) \in \pi(x)$. If $x \in V_{F'} \backslash V_\cap$, then $\deg_H(x) = \deg_{F'}(x)$. Since $x \notin V_F$ and $G' = G \backslash F$, $\deg_{G'}(x) = \deg_G(x)$. Then, by the definition of Ω', we have $\pi'(x) = \pi(x)$. Since F' is a factor of Ω', $\deg_{F'}(x) \in \pi'(x)$. Thus, $\deg_H(x) \in \pi(x)$. Now, we consider vertices in $V_\cap$. Since F and F' are edge disjoint, for every $x \in V_\cap$ we have $\deg_H(x) = \deg_F(x) + \deg_{F'}(x) \leq \deg_G(x) \leq 3$. Also, $\deg_F(x), \deg_{F'}(x) \geq 1$ since F and F' are subcubic graphs which have no isolated vertices.

- If $\deg_F(x) = 1$, then $1 \in \pi(x)$. The vertex x is 1-feasible. Thus, $\deg_G(x) \neq 2$. Since $\deg_G(x) > \deg_F(x) = 1$, $\deg_G(x) = 3$. Then, $\deg_{G'}(x) = \deg_G(x) - \deg_F(x) = 2$, $\pi'(x) = \{0, 2\}$ and $\deg_{F'}(x) = 2$.
- If $\deg_F(x) = 2$, then $\deg_G(x) = 3$ since $\deg_G(x) > \deg_F(x)$. Then, $\deg_{G'}(x) = \deg_G(x) - \deg_F(x) = 1$, $\pi'(x) = \{0, 1\}$ and $\deg_{F'}(x) = 1$.

Thus, for every $x \in V_\cap$, $\deg_H(x) = \deg_F(x) + \deg_{F'}(x) = 3 \in \pi(x)$. Thus, H is a factor of Ω.

Then, we finish the proof by a careful analysis of possible forms of F and F', and possible intersection vertices in $V_\cap$. Here, we give an example where F is a tadpole graph with a vertex u of degree 3 and a vertex v of degree 1 to illustrate this. Since $\deg_F(u) = 3$, by assumption, $\pi(u) = \{0, 2, 3\}$. Also, since $\deg_F(v) = 1 \in \pi(v)$, v is 1-feasible.

Consider possible vertices in $V_\cap$. Recall that for every $x \in V_\cap$, $\deg_F(x) = 1$ and $\deg_{F'}(x) = 2$, or $\deg_F(x) = 2$ and $\deg_{F'}(x) = 1$. Since $\deg_F(u) = 3 = \deg_G(u)$, we have $u \notin V_\cap$. Also, consider the possible forms of F'. We show that F' is not a cycle. For a contradiction, suppose that F' is a cycle. Then, all vertices of F' have degree 2. Thus, the only possible vertex in $V_\cap$ is v. If $V_\cap = \emptyset$, then for every $x \in V_{F'}$, $\deg_{F'}(x) = \deg_H(x) \in \pi(x)$. Thus, F' is a basic factor of Ω where $\omega(F') > 0$ and $\deg_{F'}(u) = 0$. We are done. Otherwise, $V_\cap = \{v\}$. Then, $\deg_F(v) = 1$ and $\deg_{F'}(v) = 2$. Since F is a tadpole graph, the graph H is a dumbbell graph where u and v are the two vertices of degree 3. Thus, H is a basic factor of Ω. Since $\omega(F') > 0$, we have $\omega(H) = \omega(F) + \omega(F') > \omega(F)$ which leads to a contraction

with F being a basic factor with the largest weight. Thus, F' is a basic factor which is not a cycle. By Definition 15, F' contains exactly two vertices of odd degree, denoted by s and t. Then, we have $V_\cap \subseteq \{v, s, t\}$.

Recall that F is a tadpole graph consisting of a path and a cycle. We use C to denote the cycle part of F, and V_C denotes its vertex set. Consider $\{s, t\} \cap V_C$. Now, we handle possible subcases according to intersection vertices appearing in V_C. There are three subcases. Below, for two points x and y, we use p_{xy} or p'_{xy} to denote a path with endpoints x and y.

1. $\{s, t\} \subseteq V_C$. Then, $\deg_F(s) = \deg_F(t) = \deg_C(s) = \deg_C(t) = 2$. In this case, $\deg_H(u) = \deg_H(s) = \deg_H(t) = 3$ and $\pi(u) = \pi(s) = \pi(t) = \{0, 2, 3\}$. Also, $\deg_{F'}(s) = \deg_{F'}(t) = 1$. Thus, F' is a path with endpoints s and t. Note that in this case, it is possible that $v \in V_{F'}$. If $v \in V_{F'}$, then $\deg_H(v) = 3$ and $\pi(v) = \{0, 1, 3\}$; otherwise, $\deg_H(v) = 1$ and $\pi(v) = \{0, 1\}$ or $\{0, 1, 3\}$. The points u, s, and t split C into three paths, p_{us}, p_{st}, p_{tu}. Then, $C = p_{us} \cup p_{st} \cup p_{tu}$. (See Figure 2.) If $\omega(C) > 0$, then we are done since C is a basic factor of Ω and $\deg_C(u) = 2$. Thus, we may assume that $\omega(C) \leq 0$.

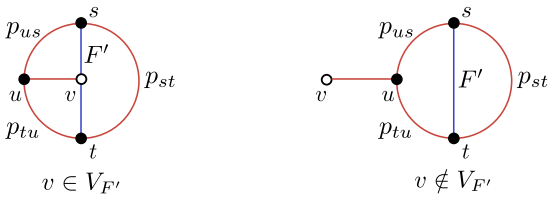


Figure 2 The two possible forms of graph H when $\{s, t\} \in V_C$. Hollow nodes denote 1-feasible vertices, and solid nodes denote 2-feasible vertices; red-colored lines denote paths in F, and blue-colored lines denote paths in F'.

Consider the graph $H_1 = H \backslash p_{st} = (F \backslash p_{st}) \cup F'$. Note that $V_{H_1} = (V_H \backslash V_{p_{st}}) \cup \{s, t\}$. For every $x \in V_{H_1} \backslash \{s, t\}$, we have $\deg_{H_1}(x) = \deg_H(x) \in \pi(x)$ since H is a factor of Ω. Also, $\deg_{H_1}(s) = 2 \in \pi(s)$ and $\deg_{H_1}(t) = 2 \in \pi(t)$. Thus, H_1 is a factor of Ω. Also, H_1 is a tadpole graph if $\deg_H(v) = 1$ or a theta graph if $\deg_H(v) = 3$. Thus, in both cases, H_1 is a *basic* factor of Ω. Since F is a basic factor of Ω with the largest weight, we have $\omega(F) \geq \omega(H_1) = \omega(F) - \omega(p_{st}) + \omega(F')$. Thus, $\omega(p_{st}) \geq \omega(F') > 0$. Since $\omega(C) = \omega(p_{st}) + \omega(p_{us}) + \omega(p_{tu}) \leq 0$, $\omega(p_{us}) + \omega(p_{tu}) < 0$. Without loss of generality, we may assume that $\omega(p_{us}) < 0$. Then, consider the graph $H_2 = H \backslash p_{us}$. Similarly, one can check that H_2 is a factor of Ω, and $\deg_{H_2}(u) = 2$. Also, H_2 is a tadpole graph if $\deg_H(v) = 1$, or a theta graph if $\deg_H(v) = 3$. Thus, H_2 is a basic factor of Ω. Moreover, $\omega(H_2) = \omega(H) - \omega(p_{us}) > 0$. We are done.
2. $\{s, t\} \cap V_C = \{s\}$ or $\{t\}$. Without loss of generality, we may assume that $s \in V_C$. Then, $\deg_H(u) = \deg_H(s) = 3$ and $\pi(u) = \pi(s) = \{0, 2, 3\}$. If $\omega(C) > 0$, then we are done since C is a basic factor of Ω and $\deg_C(u) = 2$. Thus, we may assume that $\omega(C) \leq 0$. Vertices s and u split C into two paths p_{us} and p'_{us}. Since $\omega(C) = \omega(p_{us}) + \omega(p'_{us}) \leq 0$, among them at least one is non-positive. Without loss of generality, we assume that $\omega(p_{us}) \leq 0$. Consider the graph $H' = H \backslash p_{us}$. We have $\omega(H') = \omega(H) - \omega(p_{us}) > 0$, and $\deg_{H'}(u) = 2$. Similar to the above case, one can check that H' is a factor of Ω. However, it is not clear whether H' is a *basic* factor of Ω. Consider the sub-instance $\Omega'_H = (H', \pi_{H'}, \omega_{H'})$ of Ω defined on the subgraph H' of G where $\pi_{H'}(x) = \pi(x) \cap [\deg_{H'}(x)] \subseteq \pi(x)$ for every $x \in V_{H'}$ and $\omega_{H'}$ is the restriction of ω on $E_{H'}$ (we may write $\omega_{H'}$ as ω for simplicity). Since $\omega(H') > 0$, by property 1 of Theorem 16, there is a basic factor $F^* \in \Omega_{H'}$ such that $\omega(F^*) > 0$. Then, $\deg_{F^*}(u) \in \pi_{H'}(u) = \{0, 2\}$. Now, F^* is a basic factor of Ω.

3. $\{s,t\} \cap V_C = \emptyset$. In this case, the cycle C does not intersect with F'. Then, by viewing the cycle C as an enlargement of the vertex u, this case is similar to the case that F is a path with endpoints u and v, which is proved separately. Please refer to the full paper for its proof. ◀

References

1 Jin Akiyama and Mikio Kano. *Factors and factorizations of graphs: Proof techniques in factor theory*, volume 2031. Springer, 2011.

2 Elliot Anshelevich and Adriana Karagiozova. Terminal backup, 3d matching, and covering cubic graphs. *SIAM Journal on Computing*, 40(3):678–708, 2011.

3 Richard P. Anstee. A polynomial algorithm for b-matchings: an alternative approach. *Information Processing Letters*, 24(3):153–157, 1987.

4 André Bouchet. Greedy algorithm and symmetric matroids. *Mathematical Programming*, 38(2):147–159, 1987. doi:10.1007/BF02604639.

5 André Bouchet. Matchings and Δ-matroids. *Discrete Applied Mathematics*, 24(1-3):55–62, 1989.

6 David A Cohen, Martin C Cooper, Peter G Jeavons, and Andrei A Krokhin. The complexity of soft constraint satisfaction. *Artificial Intelligence*, 170(11):983–1016, 2006.

7 Gérard Cornuéjols. General factors of graphs. *Journal of Combinatorial Theory, Series B*, 45(2):185–198, 1988. doi:10.1016/0095-8956(88)90068-8.

8 William H. Cunningham and Alfred B. Marsh. A primal algorithm for optimum matching. In *Polyhedral Combinatorics*, pages 50–72. Springer, 1978.

9 Víctor Dalmau and Daniel K. Ford. Generalized satisfability with limited occurrences per variable: A study through delta-matroid parity. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, volume 2747, pages 358–367. Springer, 2003. doi:10.1007/978-3-540-45138-9_30.

10 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM*, 61(1):1–23, 2014. doi:10.1145/2529989.

11 Szymon Dudycz and Katarzyna Paluch. Optimal general matchings. *Lecture Notes in Computer Science*, 11159 LNCS:176–189, 2018. doi:10.1007/978-3-030-00256-5_15.

12 Szymon Dudycz and Katarzyna Paluch. Optimal general matchings. *arXiv, version 3*, 2021. arXiv:1706.07418v3.

13 Zdeněk Dvořák and Martin Kupec. On Planar Boolean CSP. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP'15)*, volume 9134, pages 432–443. Springer, 2015. doi:10.1007/978-3-662-47672-735.

14 Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

15 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

16 Jack Edmonds and Ellis L Johnson. Matching: A well-solved class of integer linear programs. In *Combinatorial Structures and Their Applications*, pages 89–92. Gordon & Breach, New York, 1970.

17 Tomás Feder. Fanout limitations on constraint systems. *Theoretical Computer Science*, 255(1-2):281–293, 2001. doi:10.1016/S0304-3975(99)00288-1.

18 Tomás Feder and Daniel K. Ford. Classification of bipartite Boolean constraint satisfaction through Delta-matroid intersection. *SIAM J. Discrete Math.*, 20(2):372–394, 2006. doi:10.1137/S0895480104445009.

19 Tomás Feder and Moshe Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

20 Harold N. Gabow. *Implementation of algorithms for maximum matching on nonbipartite graphs*. PhD thesis, Stanford University, 1974.

21 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*, pages 448–456, 1983.

22 Harold N. Gabow. A scaling algorithm for weighted matching on general graphs. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science (FOCS'85)*, pages 90–100, 1985.

23 Harold N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 434–443, 1990.

24 Harold N. Gabow, Zvi Galil, and Thomas H. Spencer. Efficient implementation of graph algorithms using contraction. *Journal of the ACM (JACM)*, 36(3):540–572, 1989.

25 Harold N. Gabow and Piotr Sankowski. Algebraic algorithms for b-matching, shortest undirected paths, and f-factors. In *Proceedings of the 54th IEEE Annual Symposium on Foundations of Computer Science (FOCS'13)*, pages 137–146, 2013.

26 Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for general graph matching problems. *Journal of the ACM (JACM)*, 38(4):815–853, 1991.

27 Zvi Galil, Silvio Micali, and Harold N. Gabow. An $O(EV \setminus \log V)$ algorithm for finding a maximal weighted matching in general graphs. *SIAM Journal on Computing*, 15(1):120–130, 1986.

28 James F. Geelen, Satoru Iwata, and Kazuo Murota. The linear delta-matroid parity problem. *Journal of Combinatorial Theory, Series B*, 88(2):377–398, 2003. doi:10.1016/S0095-8956(03)00039-X.

29 Chien-Chung Huang and Telikepalli Kavitha. Efficient algorithms for maximum weight matchings in general graphs with small edge weights. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1400–1412. SIAM, 2012.

30 Gabriel Istrate. Looking for a version of Schaefer's dichotomy theorem when each variable occurs at most twice. Technical report, University of Rochester, 1997. UR CSD/TR652.

31 Aleksandr V. Karzanov. Efficient implementations of Edmonds' algorithms for finding matchings with maximum cardinality and maximum weight. *Studies in Discrete Optimization*, pages 306–327, 1976.

32 Alexandr Kazda, Vladimir Kolmogorov, and Michal Rolínek. Even delta-matroids and the complexity of planar Boolean CSPs. *ACM Transactions on Algorithms (TALG)*, 15(2):1–33, 2018.

33 Yusuke Kobayashi. Optimal general factor problem and jump system intersection. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 291–305. Springer, 2023.

34 Bernhard Korte and Jens Vygen. *Combinatorial optimization: Theory and Algorithms*, volume 21. Springer, 2018.

35 Eugene L. Lawler. *Combinatorial optimization: networks and matroids*. Holt, Reinhart and Winston, New York., 1976.

36 László Lovász. The factorization of graphs. In *Combinatorial Structures and Their Applications*, pages 243–246. Gordon & Breach, New York, 1970.

37 László Lovász. The factorization of graphs. II. *Acta Mathematica Academiae Scientiarum Hungarica*, 23(1-2):223–246, 1972. doi:10.1007/BF01889919.

38 László Lovász and Michael D. Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

39 Alfred B. Marsh. *Matching algorithms*. PhD thesis, The Johns Hopkins University, 1979.

40 Michael D. Plummer. Graph factors and factorization: 1985–2003: a survey. *Discrete Mathematics*, 307(7-8):791–821, 2007.

41 William R. Pulleyblank. *Faces of Matching Polyhedra.* PhD thesis, University of Waterloo, 1973.

42 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, 1978.

43 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

44 Shuai Shao and Stanislav Živný. A Strongly Polynomial-Time Algorithm for Weighted General Factors with Three Feasible Degrees. *arXiv*, 2023. `arXiv:2301.11761`.

45 Jácint Szabó. Good characterizations for some degree constrained subgraphs. *Journal of Combinatorial Theory, Series B*, 99(2):436–446, 2009. `doi:10.1016/J.JCTB.2008.08.009`.

46 William Thomas Tutte. The factorization of locally finite graphs. *Canadian Journal of Mathematics*, 2:44–49, 1950.

47 William Thomas Tutte. The factors of graphs. *Canadian Journal of Mathematics*, 4:314–328, 1952.

48 William Thomas Tutte. A short proof of the factor theorem for finite graphs. *Canadian Journal of mathematics*, 6:347–352, 1954.

49 Dahai Xu, Elliot Anshelevich, and Mung Chiang. On survivable access network design: Complexity and algorithms. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 186–190. IEEE, 2008.

A Δ-Matroids and Matching Realizability

A Δ-matroid is a family of sets obeying an axiom generalizing the matroid exchange axiom. Formally, a pair $M = (U, \mathcal{F})$ is a Δ-matroid if U is a finite set and $\mathcal{F}$ is a collection of subsets of U satisfying the following: for any $X, Y \in \mathcal{F}$ and any $u \in X \Delta Y$ in the symmetric difference of X and Y, there exits a $v \in X \Delta Y$ such that $X \Delta \{u, v\}$ belongs to $\mathcal{F}$ [4]. A Δ-matroid is *symmetric* if, for every pair of $X, Y \subseteq U$ with $|X| = |Y|$, we have $X \in \mathcal{F}$ if and only if $Y \in \mathcal{F}$. A Δ-matroid is *even* if for every pair of $X, Y \subseteq U$, $|X| \equiv |Y| \mod 2$.

Suppose that $U = \{u_1, u_2, \ldots, u_n\}$. A subset $V \subseteq U$ can be encoded by a binary string α_V of n-bits where the i-th bit of α_V is 1 if $u_i \in V$ and 0 if $u_i \notin V$. Then, a Δ-matroid $M = (U, \mathcal{F})$ can be represented by a relation R_M of arity $|U|$ which consists of binary strings that encode all subsets in $\mathcal{F}$. Such a representation is unique up to a permutation of variables of the relation. A degree constraint D of arity n can be viewed as an n-ary symmetric relation which consists of binary strings with the Hamming weight d for every $d \in D$. By the definition of Δ-matroids, it is easy to check that a degree constraint D (as a symmetric relation) represents a Δ-matroid if and only if D has all gaps of length at most 1.

▶ **Definition 18** (Matching Gadget). *A* gadget *using a set $\mathcal{D}$ of degree constraints consists of a graph $G = (U \cup V, E)$ where $\deg_G(u) = 1$ for every $u \in U$ and there are no edges between vertices in U, and a mapping $\pi : V \to \mathcal{D}$. A* matching gadget *is a gadget where $\mathcal{D} = \{\{0,1\}, \{1\}\}$. A degree constraint D of arity n is* matching realizable *if there exists a matching gadget $(G = (U \cup V, E), \pi : V \to \{\{0,1\}, \{1\}\})$ such that $|U| = n$ and for every $k \in [n]$, $k \in D$ if and only if for every $W \subseteq U$ with $|W| = k$, there exists a matching $F = (V_F, E_F)$ of G such that $V_F \cap U = W$ and for every $v \in V$ where $\pi(v) = \{1\}$, $v \in V_F$.*

The definition of matching realizability can be extended to a relation R of arity n by requiring the set U of n vertices in a matching gadget to represent the n variables of R. If R is realizable by a matching gadget $G = (U \cup V, E)$, then for every $\alpha \in \{0,1\}^n$, $\alpha \in R$ if and only if there is a matching $F = (V_F, E_F)$ of G such that $V_F \cap U$ is exactly the subset of U encoded by α (i.e., for every $u_i \in U$, $u_i \in V_F$ if and only if $\alpha_i = 1$), and for every $v \in V$

where $\pi(v) = \{1\}$, $v \in V_F$. Note that the matching realizability of a relation is invariant under a permutation of its variables. We say that a Δ-matroid is matching realizable if the relation representing it is matching realizable.[6]

The following result generalizes Lemma A.1 of [32].

▶ **Lemma 19.** *Suppose that $M = (U, \mathcal{F})$ is a matching realizable Δ-matroid, and $V_1, V_2 \in \mathcal{F}$. Then, $V_1 \Delta V_2$ can be partitioned into single variables $S_1, \ldots, S_k$ and pairs of variables $P_1, \ldots, P_\ell$ such that for every $P = S_{i_1} \cup \cdots \cup S_{i_r} \cup P_{j_1} \cup \cdots \cup P_{j_t}$ ($\{i_1, \ldots, i_r\} \subseteq [k], \{j_1, \ldots, j_t\} \subseteq [\ell]$), $V_1 \Delta P \in \mathcal{F}$ and $V_2 \Delta P \in \mathcal{F}$.*

▶ **Theorem 20.** *A degree constraint D of gaps of length at most 1 is matching realizable if and only if all its gaps are of the same length 0 or 1.*

Proof. By the gadget constructed in the proof of [7, Theorem 2], if a degree constraint has all gaps of length 1 then it is matching realizable.[7] We give the following gadget (Figure 3) to realize a degree constraint D with all gaps of length 0, which generalizes the gadget in [48]. Suppose that $D = \{p, p+1, \ldots, p+r\}$ of arity n where $n \geq p+r \geq p \geq 0$. Consider the following graph $G = (U \cup V, E)$: U consists of n vertices of degree 1, and V consists of two parts V_1 with $|V_1| = n$ and V_2 with $|V_2| = n - p$; the induced subgraph $G(V)$ of G induced by V is a complete bipartite graph between V_1 and V_2, and the induced subgraph $G(U \cup V_1)$ of G induced by $U \cup V_1$ is a bipartite perfect matching between U and V_1. Every vertex in V_1 is labeled by the constraint $\{1\}$. There are r vertices in V_2 labeled by $\{0, 1\}$ and the other $n - p - r$ vertices in V_2 labeled by $\{1\}$. One can check that this gadget realizes D.

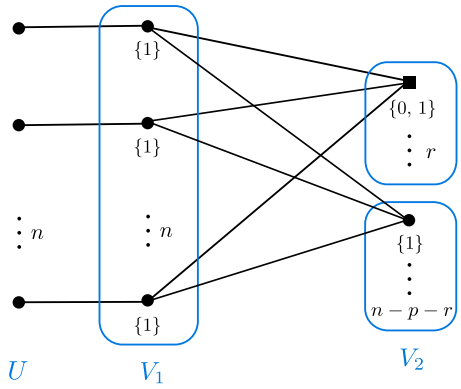


Figure 3 A matching gadget realizing $D = \{p, p+1, \ldots, p+r\}$ of arity n.

For the other direction, without loss of generality, we may assume that $\{p, p+1, p+3\} \subseteq D$ and $p+2 \notin D$. Since D has gaps of length at most 1, it can be associated with a symmetric Δ-matroid $M = (U, \mathcal{F})$. Then, there is $V_1 \in \mathcal{F}$ with $|V_1| = p$ and $V_2 \in \mathcal{F}$ with $|V_2| = p+3$. Since M is symmetric, we may pick $V_2 = V_1 \cup \{v_1, v_2, v_3\}$ for some $\{v_1, v_2, v_3\} \cap V_1 = \emptyset$. Let $S = V_1 \Delta V_2 = \{v_1, v_2, v_3\}$. By Lemma 19, S can be partitioned into single variables and/or pairs of variables such that for any union P of them, $V_2 \backslash P \in \mathcal{F}$. Since $|S| = 3$, there exists at least a single variable x_i in the partition of S such that $V_2 \backslash \{v_i\} \in \mathcal{F}$. Note that $|V_2 \backslash \{v_i\}| = p+2$. Thus, $p+2 \in D$. A contradiction. ◀

[6] This definition of matching realizability for Δ-matroids is different from the one that is usually used for even Δ-matroids [5, 13, 32], in which the gadget is only allowed to use the constraint $\{1\}$ for perfect matchings, and hence the resulting Δ-matroid must be even.

[7] We remark that [7] includes gadgets for other types of degree constraints, including type-1 and type-2, but only under a more general notion of gadget constructions that involve edges and triangles. The gadget that only involves edges is a matching gadget defined in this paper.